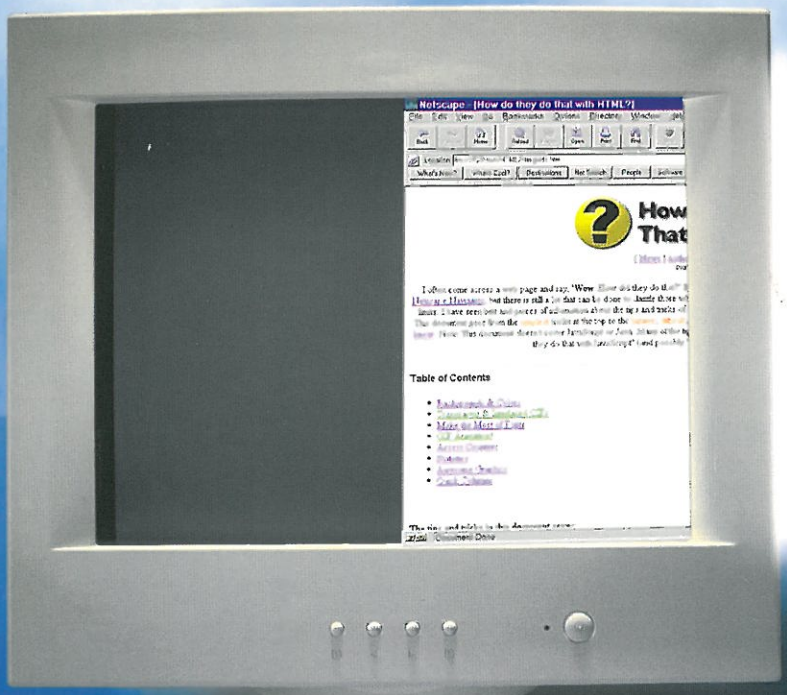
LA PRIMERA REVISTA DE PROGRAMACIÓN EN CASTELLANO

# SOLO PROGRAMADORES

AÑO III, NÚMERO 38

975 Ptas.

# NETCASTER:
## TECNOLOGÍA PUSH EN LA WEB




**CONTENIDO DEL CD-ROM**

- RealAudio 4.01
- JDK 1.1.1 para OS/2
- DirectX SDK
- ActiveMovie
- Winzip 6.3 Beta
- Controles Delphi y VBasic
- Netscape Communicator 4.02
- Service packs NT 4.0 y Visual Studio
- Visual Age for Java (entry version)
- Y mucho más

**VISUAL C++**
Transferencia de datos con las MFCs: DDX y DDV

**LINUX**
Programación X-Window: Cursor Hardware

**JAVA**
El sistema de Entrada/Salida

**INTELIGENCIA ARTIFICIAL**
Aprendizaje automático y Vida Artificial

8 413042 303299

SÓLO PROGRAMADORES

Número 38
SÓLO PROGRAMADORES
es una publicación de
TOWER COMMUNICATIONS

Director Editor
Antonio M. Ferrer Abelló
aferrer@towercom.es
Directora Adjunta
Amelia Noguera
anoguera@towercom.es

Colaboradores
E. Díaz, Xesco Hernández, Manuel de la Herrán, E. de la Lastra, Eugenio Castillo, J. Sarsa, Oscar Prieto Blanco, Ernesto Schmitz, Alejandro Reyero, Emilio Postigo, Chema Alvarez, J. Antonio Mendoza, Jorge Delgado Mendoza.

Jefe de Diseño
Fernando García Santamaría

Maquetación
Clara Francés
Tratamiento de Imagen
María Arce Giménez
Imagen de Portada
Fernando Núñez

Publicidad
Erika de la Riva (Madrid)
Tel.: (91) 661 42 11
Pepín Gallardo (Barcelona)
Tel.: (93) 213 42 29

Suscripciones
Isabel Bojo
Tel. (91) 661 42 11 Fax: (91) 661 43 86
suscrip@towercom.es

Laboratorio
Responsable Laboratorio
Javier Amado
Servicio Técnico
Oscar Casado

Filmación
Megatipo
Impresión
G. Reunidas
Distribución
SGEL

TOWER COMMUNICATIONS
Director General
Antonio M. Ferrer Abelló
Director Financiero
Francisco García Díaz de Liaño
Director de Producción
Carlos Peropadre
Directora Comercial
Carmina Ferrer
carmina@towercom.es

Redacción, Publicidad y Administración
C/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (MADRID)
Telf.: (91) 661 42 11 / Fax: (91) 661 43 86

La revista Sólo Programadores no tiene por qué estar de acuerdo con las opiniones escritas por sus colaboradores en los artículos firmados. El editor prohibe expresamente la reproducción total o parcial de los contenidos de la revista sin su autorización escrita.

Depósito legal: M-26827-1994
ISSN: 1134-4792
PRINTED IN SPAIN
COPYRIGHT 30-12-97

EDITORIAL

# Objetivos cumplidos

Los llamados profesionales del sector informático son un grupo constituido por una mezcla variopinta de personas con distintas funciones y objetivos en mente. Que si programadores junior, programadores senior, analistas-programadores, analistas funcionales, analistas orgánicos, jefes de proyectos, etcétera, etcétera, etcétera. Todos, en alguna ocasión, se preguntan cuál será el próximo lenguaje que saldrá, la herramienta más novedosa, la siguiente metodología en la que tendrán que sumergirse o, como solía decir muy a menudo – le encantaba la palabreja - un antiguo jefe de proyecto que en algún momento tuve, el siguiente paradigma que estudiar en profundidad.

Y así se pasan la vida, nos pasamos la vida: averiguando cuáles de las herramientas de desarrollo que salen al mercado serán las que tienen más futuro y por tanto más oportunidades de implantarse en el mercado, en un mercado que al final es donde todos nos lanzamos. Y aquí estamos. Aprendiendo siempre. No dejamos nunca de adquirir conocimientos. Algunos porque realmente nos gusta, nos entusiasma esto de programar; otros, porque es una forma de ganarse la vida.

En Sólo Programadores siempre pensamos que los mejores informáticos eran aquellos encuadrados dentro del primer grupo, porque no duermen, no comen, no ven la luz del sol, sólo la que despide una pantalla o un flexo, y están enamorados de su trabajo. Quizá nos equivoquemos al pensar así, tal vez lo único que tienen es una gran ventaja sobre los otros, porque su mundo laboral y privado coincide y trabajar en lo que te gusta es una gran suerte. Una inmensa suerte. Desde la redacción de la revista nos planteamos cómo ayudar a estos profesionales – a todos, los unos y los otros – a averiguar lo antes posible qué es lo último en la informática. Cuando acertamos, nos sentimos orgullosos. Si nos equivocamos, os pedimos disculpas. Siempre nos conduce un sincero interés por hacer bien nuestro trabajo.

Por eso esperamos que Sólo Programadores sea siempre otra de las herramientas que utilicéis para continuar aprendiendo. Con artículos escritos por los mejores profesionales sobre todas las áreas: la Web, lenguajes visuales, criptografía, redes, JAva, Linux, etc. Y, si queréis, podéis pedir más.

Tenéis a vuestra disposición la dirección de correo electrónico de la revista: **solop@towercom.es**, donde podéis enviar vuestras sugerencias sobre los contenidos, los temas que queráis que incluyamos y los productos que os gustaría que distribuyéramos con el CD-ROM, que todos los meses regalamos.
Sois nuestros mejores colaboradores.

# solo PROGRAMADORES

# 38


6 Noticias y libros
NOVEDADES DEL MERCADO

En esta ocasión comentamos dos clásicos del aprendizaje de la POO. Además podéis encontrar lo último que ha sucedido en el sector informático. Nuevos productos y novedades varias.

11 Redes Locales
EL DOMINIO DOMINADO

Iniciamos con este artículo una serie sobre Windows NT Server. En esta ocasión, comenzaremos con algunos conceptos básicos sobre su administración pero pretendemos llegar a ser capaces de montar una intranet y un servidor de Internet. Empezamos por el principio: los dominios.

28 Criptografía
AUTENTIFICACIÓN KERBEROS

Continuando con la serie sobre criptografía, estudiamos la forma de protegernos del acceso de personas y programas no autorizados a los recursos informáticos.

35 Programación de Juegos
LA FUNCIÓN DE EVALUACIÓN

Como los aficionados al ajedrez y la informática deben conocer, la función de evaluación es uno de los componentes claves de un programa de ajedrez.

41 Herramientas de Desarrollo
VISUAL C++ PARA WINDOWS CE

Windows CE es la propuesta de Microsoft como sistema operativo para los ordenadores de bolsillo. El llamado *Windows de mano* ya dispone de las herramientas necesarias para el desarrollo de software. En este artículo conocemos un poco más Visual C++ para Windows CE.

47
Visual C++
## TRANSFERENCIA DE DATOS CON LAS MFCS. DDX Y DDV

De forma clara, el autor de este artículo, un profesor con mucha experiencia en la enseñanza de Visual C++, nos introduce en un tema muy concreto de la programación con las MFCs, dándonos la solución para algunas situaciones en las que necesitamos escribir código de forma manual.

52
Linux
## PROGRAMACIÓN X-WINDOW: CURSOR HARDWARE

En el artículo tratamos la característica más sobresaliente de una tarjeta de vídeo moderna, el acelerador gráfico, explicando el procedimiento para añadir rutinas aceleradas, añadiendo soporte para una de las aceleraciones más extendida: el cursor hardware.

61
Inteligencia Artificial
## APRENDIZAJE AUTOMÁTICO Y VIDA ARTIFICIAL

Seguimos en nuestro empeño de crear vida y, para más dificultad, inteligente. Veremos cómo utilizar los algoritmos genéticos para crear programas que sean capaces de aprender por sí mismos a resolver todo tipo de problemas.

70
Programación en Java
## EL SISTEMA DE ENTRADA/SALIDA

En ocasiones sobran las palabras. Esta es una de ellas. Si estás interesado en este lenguaje, el artículo que nos ocupa este mes te enseñará mucho sobre la forma en que se realiza la entrada/salida, y las nuevas clases que se utilizan a partir de la aparición de la versión 1.1 del *Java Development Kit*.

22 www
## NETCASTER: INICIACIÓN A LA TECNOLOGÍA PUSH

El llamado HTML dinámico está llevándonos de la mano hacia nuevas tecnologías. En esta ocasión nos ocupamos de un nuevo paso en este camino, la propuesta de Netscape, Netcaster, que nos introduce la tecnología Push y nos sumerge en un mundo de canales y subscripciones. El resultado: la información fluye hacia nosotros más fácilmente.

81
## CONTENIDO DEL CD-ROM

Una cantidad ingente de software para trabajar en Internet y también programar, utilidades para OS/2, Windows NT, el DirectX SDK - esta vez sí- y así hasta 600 megas de contenido que estamos seguros que os será de mucha utilidad.

# Noticias

# Los desarrolladores ya pueden disponer de la última Tecnología de Reconocimiento de Voz NATURAL RECOGNITION 2.0

Se echa en falta en el mercado informático una herramienta potente para crear aplicaciones orientadas al reconocimiento de voz.

Existen algunas que son utilizadas de forma más o menos general pero, con la aparición de Natural Recognition 2.0, las empresas de este sector pueden ganar mucho. El producto constituye una plataforma de reconocimiento de voz de muy alta densidad disponible para Windows NT, Unixware y OS/2.

Soporta hasta 16 canales de reconocimiento de voz en un solo slot del PC, y además proporciona capacidad para procesar fax y llamadas. Natural Recognition se integra en CT Access, un entorno de desarrollo extensible y abierto de Natural Microsystems para la creación de funciones de miembros mixtos. CT Access proporciona un único API integrado para gestionar la interacción entre el reconocimiento de voz y el fax, la traducción de texto a voz, permitiendo una total transparencia de la tecnología que soporta la aplicación.

## Características técnicas

El reconocedor continuo soporta la entrada telefónica de dígitos de una manera continua sin producción de pitido o pausas entre las palabras, lo cual resulta de mucha utilidad en aplicaciones para la banca, por poner un ejemplo, donde ls operaciones bancarias se suelen realizar mediante la pronunciación del cliente de su cuenta bancaria y a partir de ahí se comienza con la operación seleccionada.

También soporta la entrada telefónica de un amplio abanico de palabras y frases de control para la gestión de llamadas, marcación activada por la voz, IVR y otras aplicaciones.

El producto permite además la creación de glosarios de términos personalizados que pueden ser definidos por el usuario según las necesidades de la aplicación y el ámbito en el que se utilice.

Por el momento, el producto está disponible en 30 idiomas y soporta los sistemas operativos NT, UnixWare y OS/2.

Un avance importante en el conjunto de tecnologías que se utilizan para implementa aplicaciones donde el reconocimiento de voz es primordial, podría suponer increíbles mejoras en el mundo de la empresa y los negocios, por no dejar volar la imaginación con lo que podría representar para la integración de las personas discapacitadas.

# Microsoft crea su DIVISIÓN PARA DESARROLLADORES DE SOLUCIONES

En respuesta a la demanda de aplicaciones que utilizan Microsoft Windows NT y la familia Microsoft BackOffice, el gran gigante ha anunciado la formación de su nueva división para desarrolladores, *Application Developers Customer Unit* (ADCU), que trabajará en colaboración con los desarrolladores de soluciones y los elementos componentes de la cadena de compra de aplicaciones de negocio y corporativas. La división para desarrolladores ofrecerá programas y herramientas de formación, soporte técnico, márqueting y ventas en conjunción con cada cliente, para obtener el máximo beneficio de sus productos.

Esta nueva estrategia de la organización de nuestro respetado Bill, supone un nuevo esfuerzo por dar a conocer los productos y nuevas tecnologías a los desarrolladores profesionales y corporativos además de establecer relaciones de negocio en Europa.

La ADCU está formada por un equipo de profesionales de Microsoft repartidos por todo el mundo y se encuentra trabajando ya con algunas de las empresas de desarrollo más importantes. En España, Meta 4, Logic Control, Centro de Cálculo de Sabadell, Coritel o Cap Gemini son algunas de las agraciadas.

# Asistencia Técnica a través de la red TEAMWARE LIVE HELP

La prestigiosa compañía de software TeamWARE de Fujitsu Group ha anunciado recientemente la aparición en el mercado de la versión 3.0 de su producto TeamWARE Live Help, que consitutye una importante herramienta para la asistencia técnica remota en tiempo real. El objetivo de este producto es proporcionar soporte técnico y formación on-line a los usuarios.

Gracias a TeamWARE Live Help, las personas encargadas de proporcionar soporte técnico pueden ver la pantalla del usuario, compartir el control y trabajar en su aplicación, enviar archivos y usar las herramientas que proporciona sin necesidad de desplazarse personalmente al lugar de trabajo del usuario o utilizar el teléfono.

TeamWARE Live Help 3.0 puede funcionar con usuarios que utilizan sistemas operativos como Windows NT, Windows 95 y Windows 3.1 y permite realizar conexiones sobre redes de área local, redes de área extensa, conexiones por módem y a través de Internet.

La utilización de este producto es tan sencilla como pinchar sobre el icono correspondiente para comenzar una petición al soporte técnico.

El propio usuario puede ver cómo se resuelve el problema posibilitando su aprendizaje.

## Breves

### Asforber convoca el I CERTAMEN DE PROGRAMACIÓN DE CLIPPER

La Asociación de Formación de Bergantiños convoca el certamen con los siguientes premios:

- Primer Premio: consiste en 50.000 pts. en metálico donadas por la Dirección Xeral de Política Lingüística de la Consellería de Educación Universitaria, en su afán por promover la lengua gallega dentro de la informática.
- Segundo Premio: consistirá en un lote de libros en gallego que supera las 150.000 pts. donadas por la D.X. de Promoción Cultural de la Consellería y Comunicación Social.

También se cuenta con otros premios de programas y librería donadas por programadores y empresas relacionadas con el sector. Para más información se pueden poner en contacto con ASFORBER.

### En Octubre: Conferencia Internacional de Usuarios De MACROMEDIA

Entre los días 8 y 10 de Octubre se celebrará la Conferencia Internacional de Usuarios de Macromedia en San Francisco, en el recinto Ferial Moscone Center.

Es de esperar por las novedades en el mundo de la multimedia, diseño e Internet que, la conferencia resulte muy interesante. Se tendrá la ocasión de escuchar y aprender de ponentes de muy acreditada experiencia en el sector. Los interesados pueden inscribirse a través de http://www.ucon.macromedia.com, o llamando al teléfono (800) 287-7141.

# YA está disponible la beta de Frontpage 98 Última versión de la herramienta de Creación y gestión de sitios en la Web

**Microsoft sigue apostando fuerte por la Web y la renovación de sus herramientas para el desarrollo profesional en este mundo.**

Esta nueva versión actualizada de Microsoft incorpora una serie de mejoras que facilitarán a los usuarios la construcción y el mantenimiento de los sitios Web de forma profesional. Dispone de un nuevo soporte de cuadrícula y tablas WYSIWYG, así como de más de 50 plantillas de diseño para ayudarles a aplicar de forma rápida y sencilla visualizaciones consistentes en sus sitios Web.

FrontPage 98 soporta tecnologías Web de muy reciente aparición, como el HTML dinámico, permitiendo la generación de código fuente para utilizar animaciones de texto, resúmenes dinámicos, extensiones del campo de forma, y transiciones de página.

## Tecnologías Push

También incorpora la tecnología push o definición de canales (CDF) – de lo cual hablamos en nuestro artículo de portada - mediante el asistente CDF, que habilita a los usuarios para que puedan convertir sus sitios Web favoritos en canales que serán actualizados periódicamente de forma automática una vez suscritos a ellos. Además implementa las hojas de estilo en cascada (CCS).

El rendimiento de Frontpage 98 ha sido también mejorado, ¿qué puede ser más importante que aumentar la productividad en un medio como éste?. Este incremento de productividad se ha conseguido aumentando la capacidad de publicación de contenido y permitiendo la recuperación de información en tiempo real.

Una nota importante sobre este producto es que está disponible en el CD-ROM que acompaña la revista, para que se puedan comprobar in situ todas sus *excelencias*.

# Se consolida la Asociación Nacional de Proveedores Telemáticos (ANPROTEL)

Desde que se constituyó, a finales del año pasado, se ha erigido como la plataforma que aglutina a las empresas proveedoras de servicios telemáticos de nuestro país, sin otro ánimo que el de defender el interés propio de sus asociados. Actualmente, la implantación de la Asociación en el sector es ya considerable, representando los intereses de gran parte de los proveedores españoles, ubicados a lo largo y ancho del territorio nacional.

AnProTel es una Asociacion "activa" que proporciona a sus asociados la información y los servicios necesarios para el desarrollo de su actividad, ocupándose además de tramitar los proyectos concretos que sean propuestos por los asociados y que se beneficiarán de un mayor apoyo tecnológico y económico para su consecución.

Entre los fines y objetivos de la Asociación, cabe destacar como prioritarios:

- Servir como medio de comunicación entre los diferentes proveedores de servicios telemáticos nacionales para potenciar el intercambio tecnológico y el desarrollo de proyectos de investigación conjuntos
- Prestar apoyo a sus asociados en las diferentes tareas, negociaciones, comunicaciones y proyectos que no se encuadren dentro de la explotación comercial de éstos y que supongan mejoras en los planos tecnológicos, social y ético del colectivo asociado.
- Representar, gestionar y defender los intereses profesionales colectivos de sus asociados.

Por otro lado, y para el cumplimiento de estos fines, AnProTel cuenta con una ASESORÍA JURÍDICA que cuida del estudio y la tramitación de multitud de temas, y que van, por ejemplo, desde temas de competencia desleal o abusos de posiciones dominantes hasta plagios de páginas web y temas de propiedad intelectual, habiéndose personado y ejercitado las acciones correspondientes ante los diferentes Organismos, Juzgados y Tribunales (O.C.U., Tribunal para la Defensa de la Competencia, etc., ) todo ello en defensa de los derechos y de los intereses de los asociados, y que sin duda beneficiarán al colectivo representado.

Para cualquier consulta, pueden contactar con AnProTel en la dirección de correo info@anprotel.org

# Uso de dispositivos ZyXEL, velocidades RDSI de 128K y conexiones a Infovía

Algunas de las dudas que más se les plantean a los usuarios de productos ZyXEL se refieren a la utilización de los modos de 128K que incorpora la gama RDSI, así como sus requerimientos de sistema, configuración, etc. El interés crece debido a los continuos rumores y comentarios que se reciben desde hace algún tiempo sobre la actualizacion y entrada en funcionamiento de nuevo software y equipos para INFOVIA, que permitirán la conexión a través de RDSI a velocidades mayores de 64K.

Aunque desde este medio no hemos obtenido ninguna comunicación, o información oficial por parte de la empresa responsable de dicho servicio, TELEFONICA, pruebas realizadas confirman que ha sido posible establecer comunicaciones a través de Infovía en modo de 128K, siguiendo el estándar PPP Multilink y la agregación de canales.

Así mismo, es muy importante tener en cuenta que dicho modo siempre es posible activarlo en llamadas directas entre dispositivos RDSI (por ejemplo, entre dos routers Prestige), a proveedores de servicios de información, Internet o acceso remoto, si en sus equipos tienen contemplada y permitida tal posibilidad.

# NETSHOW 2.0, EL SERVIDOR MULTIMEDIA DE MICROSOFT está disponible en la red para usuarios de Windows NT Server

El servidor multimedia de Microsoft ya está disponible en la red de forma gratuita hasta diciembre de este año.Los usuarios de Windows NT Server podrán descargarlo sin coste alguno. Para aquellos que no conozcan este producto de Microsoft, NetShow es la tecnología desarrollada para producir contenidos multimedia dotados de vídeo y audio. Además permite generar audio con imágenes sincronizadas sobre Internet e intranets corporativas.

NetShow 2.0 ha tenido una gran aceptación dentro del sector multimedia, siempre según datos de Microsoft. La devolución de HongKong a China ha sido retransmitido en directo a través de la red y un producto de estas características puede ser de mucha utilidad para las emisoras de radio que trabajen a través de Internet. El que los usuarios puedan emitir contenidos dotados de audio y vídeo en tiempo real supone una gran ventaja en medios de comunicación y se prevee que sea ampliamente utilizado también por desarrolladores de Internet, lo cual amplia mucho el mercado al que se destina.

La arquitectura abierta NetShow es muy sencillo de utilizar y permite una calidad muy elevada al ser una plataforma basada en estándares que incluye más de 20 codificadores de audio y vídeo, así como comprensión de modelos, para conseguir llegar a todos los proveedores de contenido. Soporta una amplia variedad de protocolos de transporte a través de la red y limita la necesidad habitual de anchos de banda muy grandes.

Para más información sobre Microsoft puede visitar sus páginas en Internet en http://www.microsoft.com/spain/.

## Breves

### CONTROLES ACTIVEX PARA INTEGRAR DATOS DE HOST EN APLICACIONES OO

Con la versión 6.3 de EXTRA! Personal Client (EPC), que incluye los EXTRA! Objects, los desarrolladores podrían integrar los datos y la información del host en nuevas aplicaciones orientadas a objetos, mediante la inclusión de un conjunto de ActiveX.

Además, Attachmate, el padre de la criatura, ha lanzado el EXTRA! Objects SDK que, como su nombre parece decir, es un interfaz de programación para utilizar los componentes de EPC mediante un conjunto de APIs abiertas con capacidad de automatización.

### ORACLE 8 SERÁ COMPATIBLE CON EL SOFTWARE DE BAAN COMPANY

Baan Company es un proveedor de aplicaciones de gestión empresarial integrada que tiene como campo de acción numerosas empresas distribuidas por todo el mundo.

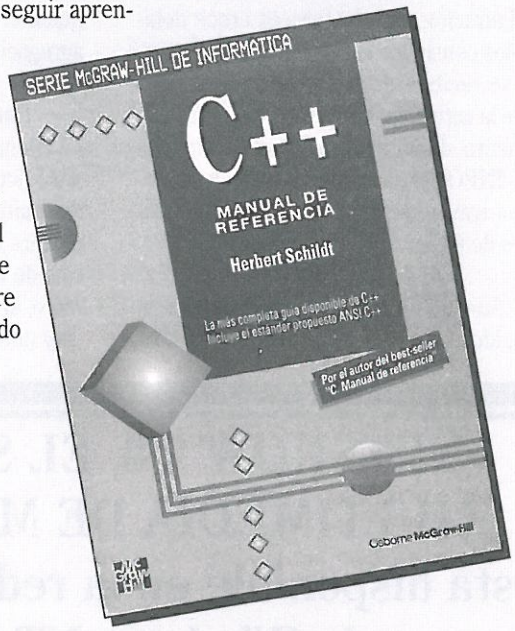
Anunció tiempo atrás que ofercería su apoyo a Oracle 8, la BAse de DAtos para NEtwork Computing que gestiona en red grandes cantidades de información, permitiendo transacciones a alta velocidad, y sofisticadas aplicaciones de objetos relacionales. Baan IV está a punto de soportar Oracle 8, ya que su aparición se planea para este cercano otoño.

Los clientes se beneficiarán de las prestaciones de Proceso de Transacciones en Línea (OLTP), que permitirá a miles de usuarios concurrentes de BAAN IV acceder a la ingente cantidad de datos que gestiona la BD.

# LIBROS

## C++ Manual de Referencia

Hay ocasiones en las que los auténticos protagonistas son los que llevan siéndolo durante años. Este libro es un clásico, pero no por ello deja de tener actualidad. Muchas son las empresas que en España siguen desarrollando con C++, y muchos son los buenos programadores que utilizan esta obra como guía para seguir aprendiendo, recordar lo que ya saben o resolver alguna duda que les surge en un momento de agobio y desazón, cuando, delante de un compilador que no quiere dar el mensaje de todo OK, quieren comprobar si algo que han dado por correcto, no lo es tanto y rectificar.

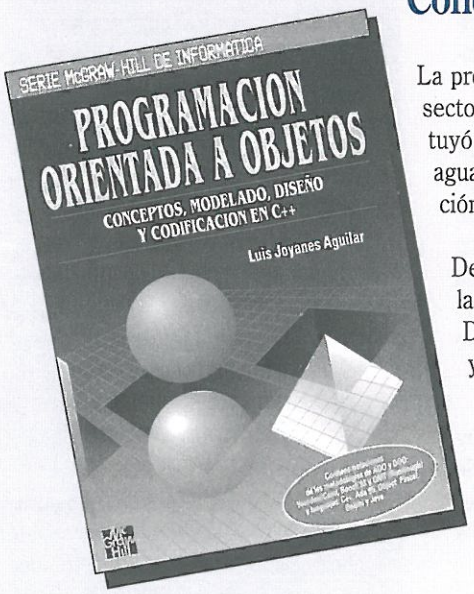



Herbert Schildt es un viejo conocido de los programadores de C, autor de conocidas obras sobre este conocido lenguaje, con ésta sigue dando en el clavo explicando aquellos aspectos de la programación en C++ que difieren de la de C. Un gran acierto es el comentar de forma explícita las diferencias entre ambos, sin que llegue a resultar agobiante para un experto en C. Sigue estando en el candelero. Una obra excelente.

Editorial: Osborne/McGraw-Hill
Autor: Herbert Schildt
Idioma: castellano
Precio: 5.345 pts.
Número de páginas: 592
Valoración: Muy bueno

## Programación orientada a objetos

### Conceptos, Modelado, Diseño y Codificación en C++




La programación orientada a objetos sigue dando guerra. Ninguna de las empresas del sector informático que se precie abandona este paradigma de programación que constituyó la esperanza para aquellos que empezaban a ver que muchos proyectos hacían aguas debido a la dificultad de ampliarlos siguiendo las metodologías de la programación estructural.

De una forma sencilla y directa podemos entender los conceptos más importantes de la POO. En esta obra, sin embargo, se echa de menos algún capítulo más sobre el Diseño en la programación orientada a objetos, la parte más complicada en un proyecto informático.

Editorial: McGraw-Hill
Autor: Luis Joyanes Aguilar
Idioma: castellano
Precio: 4.200 pts.
Número de páginas: 688
Valoración: Bueno

# El dominio dominado

*Enrique Díaz*

Una buena comprensión de los componentes de un dominio y de la forma en que se interrelacionan es muy importante a la hora de tomar decisiones en la implantación de una estructura de dominios, así como para utilizar adecuadamente las funcionalidades del servicio de directorios de Windows NT Server.

*Conocer los componentes de un dominio y la forma en que se interrelacionan es importante en la implantación de una estructura de dominios*

Hace algunos años las redes locales raramente disponían de más de dos servidores, con lo que encontrar algún recurso en la red era facilísimo. Y la administración tampoco planteaba dificultad alguna. Sin embargo, al ir añadiendo servidores la cosa se complicaba notablemente; el usuario debía saber en qué servidor se encontraba, por ejemplo, el servidor de impresión, dónde el motor de base de datos, dónde el servidor de ficheros, etc.

Para los administradores era mucho más duro. Dar de alta a un usuario con acceso a toda la red suponía crear tantas cuentas de usuario como servidores existieran en la red. Y todo esto sin disponer de ninguna utilidad para ayudarte en la administración de los múltiples servidores.

Para solucionar estos problemas surgió el concepto de servicios de directorio, que permiten al usuario conectarse a la red y tener acceso a los recursos de la misma sin preocuparse del servidor en que se encuentran esos recursos. El usuario no necesita conocer a qué servidor se conecta, ya que sencillamente se conecta a la red, y a partir de ahí tendrá acceso a los recursos de la red.

Sin embargo, esta maravilla de la técnica requiere una cuidadosa planificación de la red y que todos los departamentos se involucren en su diseño y administración. También se necesitará que los administradores sean personal más cualificado que el requerido para sistemas de red que no soportan servicios de directorio.

Una red con una mala planificación tiende a descontrolar, es decir, todo se conecta con todo, y todo el mundo tiene acceso y derechos sobre todo, con lo cual empiezan a surgir problemas con la seguridad y el adecuado uso de la red.

Los servicios de directorio proporcionan de esta forma un eficaz sistema de acceso a la red que requiere una única autentificación ante el servicio de directorios, proporcionando al mismo tiempo a los administradores un punto

Iniciamos con este artículo una serie sobre Windows NT Server. Empezaremos con algunos conceptos sobre su administración para acabar montando una intranet y un centro servidor de Internet. De momento, comenzamos por el principio; veremos ahora los dominios, un concepto básico si se desea entender las redes bajo Windows NT.

de administración centralizada de los recursos distribuidos en la red.

## El dominio

El dominio es la unidad administrativa de los servicios de directorio de Windows NT; consiste en una agrupación lógica de servidores y otros miembros del dominio que comparten seguridad e información de usuario comunes. El administrador crea las cuentas de usuario dentro de los dominios. Estas cuentas incluyen información del usuario, pertenencia a grupos e información sobre la seguridad, entre otras.

Los dominios no significan proximidad física o una configuración específica de red. Un dominio permite que los recursos de ordenadores desperdigados por el mundo entero nos resulten tan locales como los del ordenador de la mesa de al lado.

El dominio viene a ser una buena combinación de los conceptos de grupo de trabajo y servicios de directorio:

El grupo de trabajo (como los de Microsoft para Trabajo en Grupo) consiste en una serie de usuarios que comparten los recursos de sus ordenadores, como módems, unidades de disco compacto, etc. Sin embargo, este concepto presenta los mismos problemas para localizar los recursos que comentábamos antes para las redes con varios servidores; además, cada usuario administra su ordenador y, por tanto, hace lo que le da la gana.

En estos grupos de trabajo es frecuente encontrarse con que un recurso ha desaparecido y tener que ir a Menganito (que es muy celoso del tiempo de proceso de su ordenador) para decirle que vuelva a compartir el directorio.

Los dominios pueden ser desarrollados fácilmente y con menos planificación que un servicio de directorio y, al igual que éstos, un dominio coloca los recursos de varios servidores en una única estructura organizativa y, del mismo modo que con un servicio de directorio, los usuarios se conectan no a un servidor, sino a un dominio.

Así pues podemos localizar los recursos de la red mejor de lo que se haría en un grupo de trabajo, con una administración centralizada —o no, pero eso lo veremos más tarde y al menos sí será jerarquizada– y un buen nivel de seguridad. También es posible crear varios dominios y establecer relaciones de confianza ellos. La relación de confianza consiste sencillamente en que un dominio confía en que el usuario autentificó correctamente su conexión en otro dominio que no es el suyo, y por tanto, le da acceso. Estas relaciones de confianza permiten que un usuario conectado a un dominio pueda acceder a otros dominios que confían en el dominio al que se conectó el usuario.

Los servidores pueden actuar de tres maneras diferentes en un dominio, a saber, como PDC, BDC o servidor independiente.

*Los servicios de directorio proporcionan un punto de conexión y administración centralizados*

Un PDC (*Primary Domain Controller,* Controlador Principal de Dominio) es el servidor donde se almacena la copia maestra de la base de datos del directorio que contiene información sobre los grupos y usuarios del dominio. El PDC registra todos los cambios que se realizan en la base de datos y es el único que recibe estos cambios, por tanto sólo puede existir un único PDC por cada dominio.

El BDC (*Backup Domain Controller,* Controlador de Reserva de Dominio) mantiene una copia de la base de datos del directorio que se sincroniza automáticamente de forma periódica. Cuando se realiza algún cambio en la base de datos del directorio (como creación o modificación de cuentas de usuario y asignaciones de seguridad), el PDC recibe los cambios, y cuando un BDC lo requiere se le pasan todos los cambios habidos desde la última sincronización. A esto se le llama *sincronización parcial*.

Los cambios en la base de datos del directorio se almacenan en el *registro de cambios,* el cual tiene limitada su capacidad de almacenar datos. El tamaño del registro de cambios determina cuánto tiempo se guardan los cambios. Así pues, en el registro los cambios mas antiguos se borran para dejar lugar a los nuevos. Si un BDC ha estado desconectado o no solicita una sincronización durante el tiempo suficiente como para perder cambios que ya han sido borrados de la base de datos, se produce una *sincronización completa;* se le pasa una copia completa de la base de datos del dominio.

Esto, obviamente, también sucede cuando se agrega un nuevo BDC al dominio. Las sincronizaciones se producen cada cinco minutos, pero este parámetro se puede variar a través del *registro*, al igual que el tamaño de la base de datos del directorio. Normalmente no es necesario cambiar estos valores, ya que el registro soporta unos 2.000 cambios y sólo en redes muy grandes se puede llegar a realizar tantos cambios en cinco minutos. En ocasiones podemos forzar una sincronización inmediata, si por ejemplo creamos una cuenta de usuario y necesitamos que ese usuario tenga acceso a todos los recursos de la red rápidamente. De otro modo habría que esperar los cinco minutos de rigor para que el resto de servidores sincronicen la base de datos y autoricen al usuario para utilizar sus recursos.

Si por cualquier motivo no funciona o tenemos que desconectar un controlador principal de dominio, podremos convertir un controlador de reserva en controlador principal (más adelante veremos cómo).

El tercer modo en que un servidor puede actuar en un dominio es como servidor independiente. Los servidores independientes no participan en la gestión de la base de datos y por lo tanto no reciben copias de la base de datos ni autentifican cuentas del directorio; pueden participar (o no) en el dominio. Los servidores independientes son útiles para realizar tareas en las que el tiempo de respuesta es crítico o soportar aplicaciones muy 'pesadas' como Microsoft SQL Server. Tendremos además la ventaja de que podemos moverlos de un dominio a otro sin ningún problema.

*Los servidores independientes no participan en la gestión de la base de datos*

También soportan las siguientes características especiales:

- Hasta 256 conexiones simultáneas a través del servicio de acceso remoto.
- Tolerancia a fallos avanzada (espejo y duplicación de disco, RAID 5).
- Acceso de Macintosh a los Servicios de archivo e impresión de Windows NT Server.
- Soporte de servidor de inicio remoto para clientes MS-DOS y Windows 3.x.

## Las relaciones de confianza

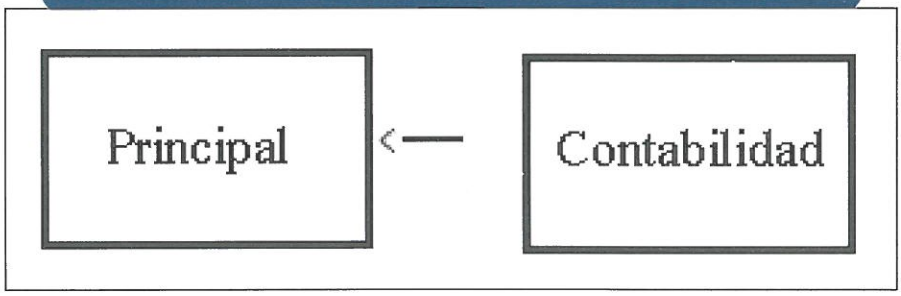
En empresas pequeñas, nos bastará con crear un único dominio, pero en empresas un poco mayores, quizá nos interese establecer varios dominios. Lo que suele hacerse es tener un dominio que almacena todas las cuentas de usuario y establecer relaciones de confianza con el resto de los dominios. Por otra parte, el tamaño de la base de datos del dominio no debe exceder de 40 MB, lo que nos limita el número de usuarios, grupos y estaciones de trabajo que pueden definirse en un dominio. En cualquier caso, el hecho de acercarnos siquiera a semejante cifra es síntoma de que debemos organizar una administración estructurada y delegar parte de la responsabilidad en diferentes dominios.

En la figura 1 se ha establecido una relación de confianza entre el dominio Contabilidad y el dominio Principal. De este modo se indica que Contabilidad confía en que Principal ha validado correctamente la conexión del usuario y le da acceso. Esta relación únicamente funciona en esta dirección; es decir, Contabilidad confía en Principal, pero Principal no confía en Contabilidad.

Así, los recursos de Contabilidad, el *dominio que confía,* están disponibles para los usuarios que se conecten a Principal, *el dominio en el que se confía.*

Para que la relación de confianza funcione en ambos sentidos tendremos que añadir una relación de confianza entre Principal y Contabilidad (figura 2).

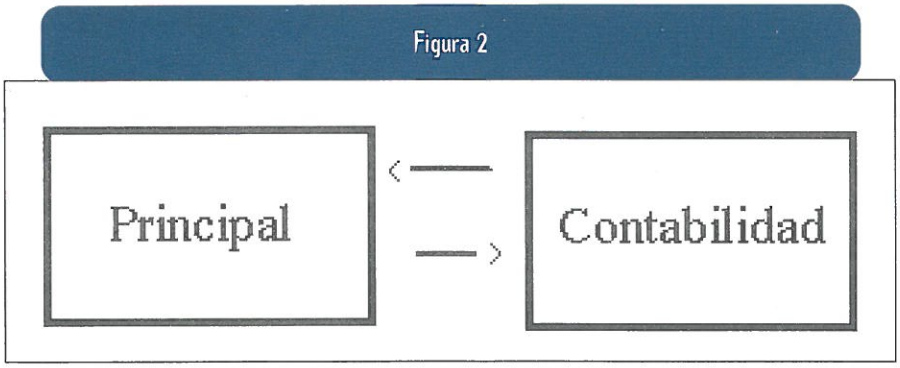
Figura 1




De este modo un usuario que se conecte a cualquiera de los dos dominios podrá utilizar los recursos de ambos. Este tipo de relación de confianza bidireccional permite que cada dominio tenga sus propias cuentas y recursos; algo que resulta de mucha utilidad si no queremos tener una administración totalmente centralizada.

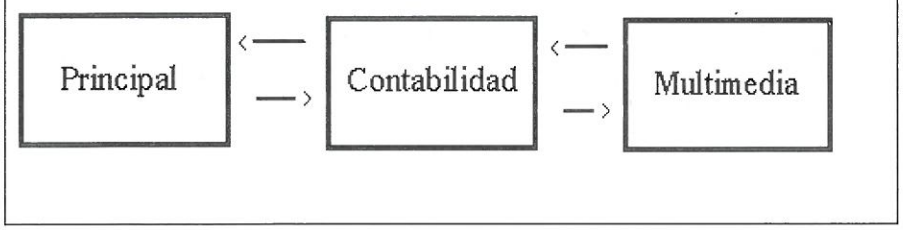
Es necesario tener muy claro que una relación de confianza bidireccional no significa que cualquiera se conecta a cualquier dominio. Cada cual se conecta al dominio donde tenga una cuenta de usuario; y a partir de ahí los dominios se las apañan con sus relaciones de confianza.

Figura 2



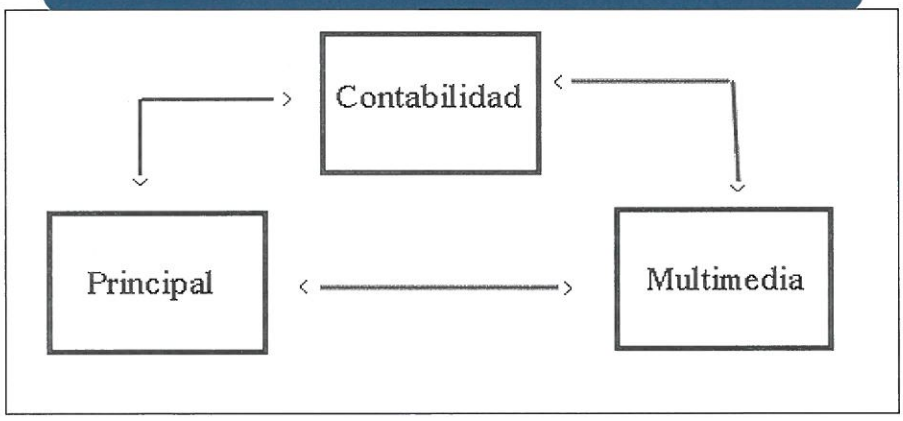

Los dominios son de naturaleza poco confianzuda, de modo que nunca hacen migas con los amigos de los amigos. Dicho de manera más técnica: las relaciones de confianza no son transitivas. En la figura 3 existe una relación bidireccional entre Contabilidad y

Figura 3




Principal y entre Multimedia y Principal. Aunque Principal confía en Contabilidad y este confía a su vez en Multimedia, no por eso existe relación de confianza entre Principal y Multimedia. Para conseguir establecer una relación de confianza entre todos ellos lo haremos estableciendo relaciones bidireccionales entre todos y cada uno de los dominios, como se puede observar en la figura 4.

Figura 4




Para establecer las relaciones de confianza entre dos dominios debemos configurar ambos dominios; uno de ellos debe ser configurado para que el otro confíe en él. A continuación configuraremos el segundo dominio para que confíe en el primero.

Hagamos por ejemplo que el dominio *Mulimedia* confíe en *Principal*. Para ello arrancaremos el *Administrador de Usuarios para Dominios* del PDC del dominio *Principal*. Dentro del menú *Directivas* elegimos la opción *Relaciones de confianza*. Pinchamos el botón *Agregar* en la parte correspondiente a *Dominios de confianza* (figura 7); en la caja de texto *Dominio que confía* tecleamos el nombre del dominio que confiará en el que estamos configurando, en nuestro caso el nombre de dominio a teclear será *Multimedia*. De forma opcional podemos establecer una contraseña que nos será requerida cuando completemos el establecimiento de la relación de confianza. Una vez que hayamos finalizado, el dominio *Multimedia* se habrá añadido a la lista de dominios de confianza.

*Los amigos de tus amigos no son mis amigos: las relaciones de confianza no son transitivas*

A continuación arrancaremos el *Administrador de Usuarios para Dominios* del PDC que confiará, esto es, *Multimedia*. Procedemos de igual manera que en el caso anterior, pero en esta ocasión pinchamos en el botón *Agregar* de la parte correspondiente a *Dominio en los que se confía*. Ahora tecleamos el nombre del dominio en el cual confiará el dominio que nos encontramos configurando, en nuestro ejemplo sería *Principal*.

Si especificamos contraseña al configurar el dominio de confianza, deberemos ponerla ahora. Cuando finalicemos, pulsamos *Aceptar* y aparecerá un mensaje indicando que se ha establecido la relación de confianza, o bien un mensaje de error.

Para conseguir eliminar las relaciones de confianza obraremos a la inversa, es decir, con la opción *Quitar* que ahora tendremos disponible.

Tabla 1. Tamaño de los objetos del domino

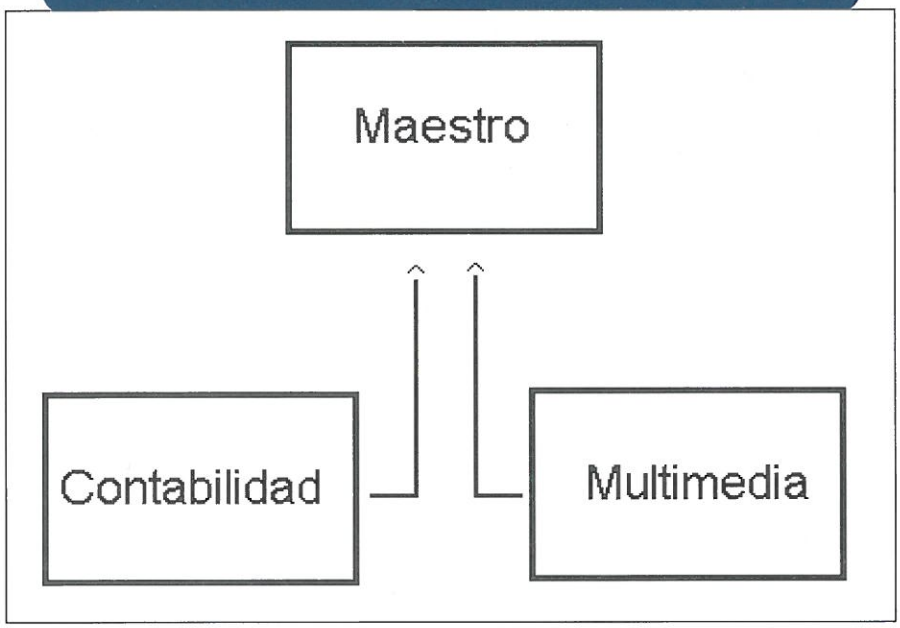
| Objeto | Tamaño |
|---|---|
| Cuenta de usuario | 1,0K |
| Cuenta de equipo | 0,5K |
| Cuenta de grupo (300 miembros) | 4,0K |

## Modelos de dominio

Los modelos de dominio se utilizan para agrupar dominios con vínculos de comunicación y administración para administrar tanto usuarios como recursos.

Un sólo dominio puede soportar unos 26.000 usuarios y 250 grupos, así pues, con un sólo dominio podremos manejar casi cualquier red. Sin embargo este límite no es la única razón para la creación de nuevos dominios. Podemos

Figura 5




querer hacerlo también por razones organizativas.

El límite del tamaño de un dominio está realmente en el tamaño de la base de datos de directorio, que no debe pasar de 40 MB. Un dominio consta de cuentas de usuario, de equipo y cuentas de grupo. Cada uno de estos objetos ocupa espacio en la base de datos de directorio (Tabla 1). Atendiendo a la cantidad de memoria que ocupa cada objeto, podemos tener configuraciones que superen los 26.000 usuarios mencionados anteriormente. Por ejemplo, si creamos exclusivamente cuentas de usuario y no tenemos ni cuentas de equipo ni grupos, la base de datos soportará 40.000 cuentas de usuario.

También es posible saber cuál es el tamaño actual de nuestra base de datos: un dominio con dos mil usuarios, dos mil cuentas de equipo y 30 grupos tendrá una base de datos de 3,12 MB.

Existen cuatro modelos de dominio con relaciones de confianza, de manera que con una única conexión se pueda acceder a todos los recursos:

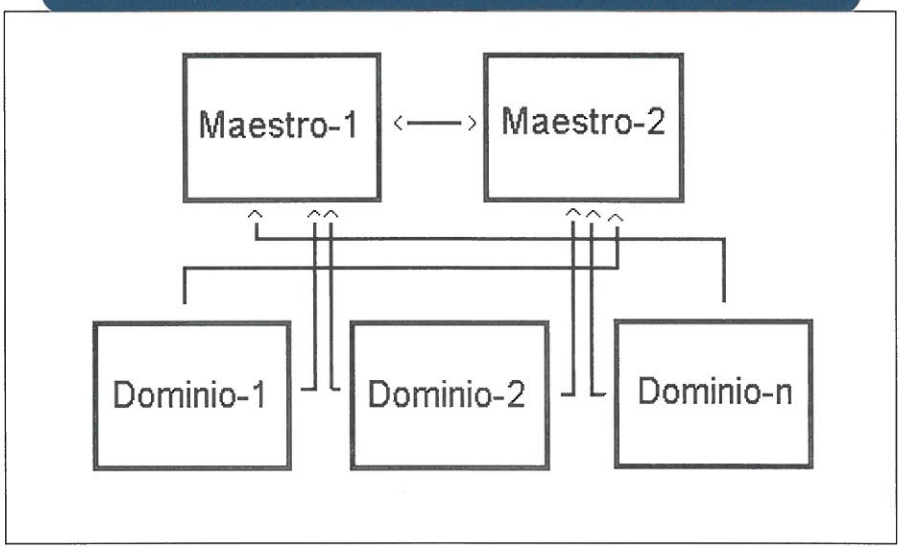
- Dominio sencillo.
- Dominio Maestro.
- Dominio de múltiples maestros.
- Confianza completa.

El *dominio sencillo* consiste en un único dominio al que pertenecen todos los servidores y todas las cuentas de usuario. Tiene la ventaja de ser muy fácil de administrar y tener administración centralizada, pero el rendimiento de la red decrecerá si el dominio llega a ser demasiado grande, bien por número de usuarios, de servidores o por el tamaño de la propia red. Además, no permite una buena estructuración ya que no podremos tener un dominio distinto en cada departamento.

*El dominio sencillo es un dominio al que pertenecen todos los servidores y las cuentas de usuario*

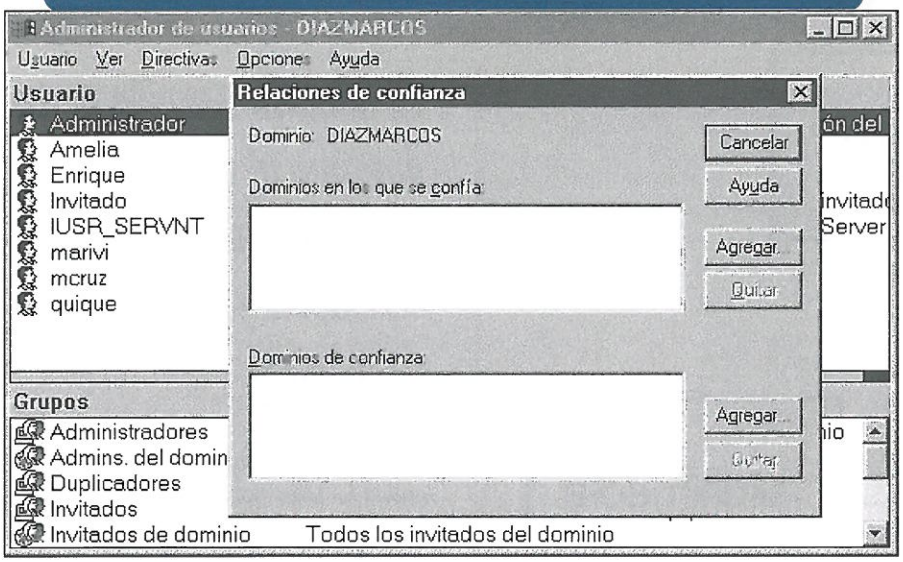
El *dominio maestro* consiste en designar un único dominio donde se administran todas las cuentas de usuario y grupos globales. El resto de dominios confían en el dominio maestro y tienen definidos grupos locales y los recursos de la red. Este modelo es la mejor opción si tenemos una red no muy grande y queremos fragmentarla por motivos organizativos.

Figura 6




En el ejemplo de la figura 5 los usuarios se conectan al dominio Maestro y obtienen los recursos de los dominios Contabilidad y Multimedia, que confían en Maestro. Este modelo nos ofrece una administración centralizada de la seguridad y tener un responsable que administre el dominio maestro (y por tanto, toda la red),

Figura 7




mientras que operadores con menores conocimientos pueden administrar el resto de dominios. Asimismo distribuiremos la actividad de exploración de la red entre los dominios de los departamentos, que pueden administrar sus propios dominios y organizar los recursos de forma lógica. Por otra parte, añadir dominios es muy fácil ya que sólo se establece una relación de confianza unidireccional entre el nuevo dominio y el maestro.

*Los dominios que no son maestros pueden establecer relaciones de confianza entre sí*

Hay que recordar aquí el límite de 40 MB de tamaño máximo que es capaz de soportar la base de datos del controlador principal de dominio. Sin embargo, lo más seguro es que el rendimiento de la red caiga antes de llegar a ese límite, debido a la gran cantidad de usuarios que se conectan al dominio maestro.

Como ampliación a este modelo podemos desarrollar el de *múltiples maestros*. Consiste en añadir dominios maestros y establecer relaciones de confianza bidireccionales entre todos los dominios maestros, mientras que el resto de dominios establece una relación de confianza unidireccional con todos los maestros.

En la figura 6 vemos las relaciones de confianza que se han de establecer. Los dominios que no son maestros pueden establecer relaciones de confianza entre sí, pero esto no es necesario. De esta manera los dominios maestros soportan las cuentas de usuario y de grupo, procurando que se las repartan por igual, más o menos. También es aconsejable que los usuarios que tengan alguna relación se conecten al mismo maestro. Este modelo es teóricamente escalable hasta el infinito, pero debemos tener en cuenta que cuantos más dominios maestros agreguemos más complejas serán las relaciones de confianza.

Los modelos con dominio maestro, ya sea sencillo o múltiple, asumen una administración centralizada. Si deseamos una administración descentralizada, al estilo del modelo autonómico español, podemos establecer un modelo de confianza completa, esto es, que cada uno de los dominios confía en el resto.

De esta manera no necesitamos un departamento que se ocupe de la administración, ya que aquí cada palo aguanta su vela, pero a cambio necesitamos crear una gran cantidad de relaciones de confianza y la red es interdependiente de todos sus dominios para su administración.

## Administración de dominios

Los controladores principal y de reserva de dominio únicamente pueden crearse durante el proceso de instalación, al indicar la función que realizará el servidor. Sin embargo, es posible promover y degradar controladores de dominio o cambiarlos de dominio.

A la hora de administrar dominios debemos tener en cuenta unas reglas muy básicas:

- • Cada dominio ha de tener un nombre único y no puede contener espacios. En cada dominio sólo puede haber un controlador principal, y por tanto no puede añadirse un controlador principal a un dominio ya existente que ya tenga PDC, lo que sí podemos hacer es añadir un controlador de reserva, siempre y cuando exista un controlador principal activo en ese momento.

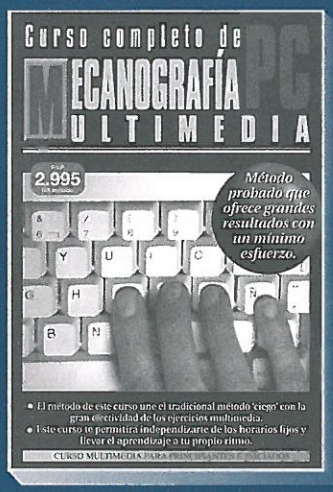
- • Un controlador de reserva se puede añadir a un dominio durante la instalación o bien utilizando el administrador de servidores.

*No podremos cambiar un BDC de dominio mientras exista un PDC activo en su dominio actual*

# Curso completo de MECANOGRAFÍA PC MULTIMEDIA

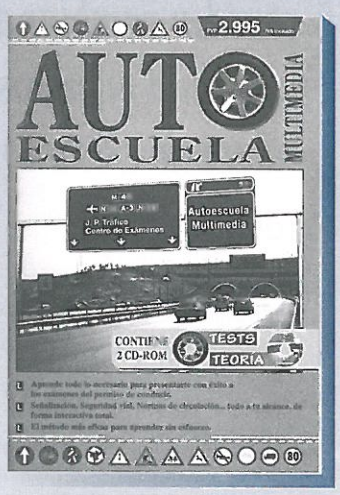
P.V.P
**2.995**
**Ptas. c.u.**
IVA Incluido

- El método de este curso une el tradicional método 'ciego' con la gran efectividad de los ejercicios multimedia.
- Este curso te permitirá independizarte de los horarios fijos y llevar el aprendizaje a tu propio ritmo.




# AUTO ESCUELA MULTIMEDIA

- Aprende todo lo necesario para presentarte con éxito a los exámenes del permiso de conducir.
- Señalización, Seguridad vial, Normas de circulación... todo a tu alcance, de forma interactiva total.
- El método más eficaz para aprender sin esfuerzo.




ABETO EDITORIAL

Envíe este cupón por correo o fax (91) 661 43 86 o llamando al teléfono: (91) 661 42 11* de lunes a jueves de 9:00 a 14:00 y de 15:00 a 18:00 h., y viernes de 9:00 a 15:00 h.

**Deseo que me envíen:** ☐ AUTOESCUELA MULTIMEDIA **por 2.995 ptas. + 250 de gastos de envío**
☐ MECANOGRAFÍA PC MULTIMEDIA **por 2.995 ptas. + 250 de gastos de envío**

Nombre y apellidos ..... Domilicio .....
Población ..... C.P. ..... Provincia .....
Teléfono ..... Edad ..... Profesión .....

**FORMA DE PAGO:**

☐ Talón a ABETO EDITORIAL ☐ Giro Postal (adjunto fotocopia de resguardo)
☐ Contra-reembolso ☐ VISA nº _____ _____ _____ _____ Cad. __ __

ABETO EDITORIAL
C/ Aragoneses, 7
28108 Alcobendas (Madrid)

Los dominios se pueden crear mediante la instalación de un servidor como controlador principal, o bien cambiando un controlador principal de dominio ya existente. Veamos cómo hacerlo.

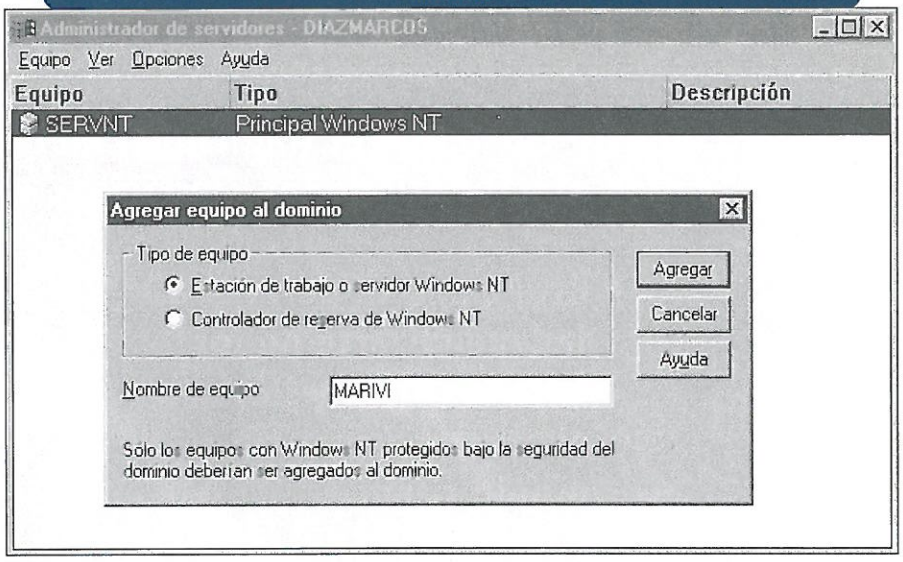
Para ello pincharemos dos veces sobre el icono red del servidor PDC y seleccionamos la solapa *Identificación*. Nos aparecerá el nombre del equipo y de dominio. Pulsando sombre el botón *cambiar* podremos especificar el dominio al que deseamos cambiar al PDC. El nombre de dominio que escribamos no puede existir en la red, de modo que lo que estamos haciendo realmente es crear un nuevo dominio. Pulsamos aceptar y nos aparecerá un mensaje indicándonos que también deberemos cambiarlo en todos los servidores y estaciones de trabajo pertenecientes al dominio, y también restablecer todas las relaciones de confianza.

*Los servidores BDC no se pueden cambiar de dominio mientras en su dominio exista un PDC activo*

Para hacer efectivos los cambios, reiniciamos el servidor. De este modo tendremos que el anterior dominio se encuentra en estos momentos sin un PDC, lo cual nos viene estupendamente para seguir trabajando ya que los servidores BDC no se pueden cambiar de dominio mientras en su dominio exista un PDC activo. Así pues este es el momento ideal para pasar los BDC que deseemos al nuevo dominio.

Lo veremos mejor con un ejemplo: supongamos que tenemos un dominio *Principal* cuyo servidor PDC se llama *Principal-1,* y dos servidores BDC cuyos nombres son *Principal-2* y *Principal-3*. Cambiamos el servidor *Principal-1* al nuevo dominio *Contabilidad* del modo que hemos explicado antes. Cuando reiniciamos tenemos un nuevo el nuevo dominio *Contabilidad,* y el servidor *Principal-1* es su PDC. Justo ahora que tenemos al dominio *Principal* sin un PDC podremos cambiar el servidor *Principal-2* al nuevo dominio. Reiniciamos *Principal-2* y ya le tendremos como BDC de *Contabilidad.* El orden a seguir es muy importante; de otro modo se producirían errores. Si en el ejemplo anterior cambiamos a *Principal-1,* luego promovemos a *Principal-3* y finalmente tratásemos de cambiar de dominio a *Principal-2,* nos encontraríamos con que no podemos hacerlo, ya que el dominio *Principal* ya dispone de PDC. Así pues, para poder cambiar a *Principal-2* al nuevo dominio, tendríamos que repetir el proceso de instalación y ponerlo en el nuevo domino *Contabilidad.* Por otra parte, los servidores independientes pueden ser cambiados de un dominio a otro sin problemas.

Figura 8. Mediante el Administrador de servidores podemos realizar la gestión de los dominios.




## Promover y degradar controladores

No sólo al crear nuevos dominios se hace necesario promover o degradar un controlador de dominio. Tendremos que hacerlo cuando necesitemos desconectar el PDC para realizar operaciones de mantenimiento, o sencillamente si se estropea. En el caso de que fuera por poco tiempo, podríamos pasar sin PDC y no será necesario hacer nada, pero no podremos realizar cambios en la base de datos del directorio.

Si promocionamos un BDC mientras el PDC está activo, el PDC pasará una copia de la base de datos al BDC que estamos promocionando y acto seguido se degradará al puesto de BDC. Si el PDC no está activo cuando promocionamos un BDC, su base de datos pasará a considerarse como la base de datos principal, de modo que se perderán todos los cambios que no fueron copiados desde la última sincronización. Cuando tengamos el antiguo PDC disponible, debemos degradarlo a BDC.

Para promover un servidor a PDC cerraremos primero todas las conexiones de los usuarios y arrancaremos la utilidad *Administrador de Servidores.* Una vez allí, seleccionaremos en la lista el servidor que queremos promover y en el menú *Equipo* seleccionamos la opción *Promover a controlador principal de dominio*. Aparecerá entonces un aviso indicándonos que debemos cerrar todas las conexiones de usuarios. Marcamos *Sí* y de esta forma la promoción queda realizada.

Si hemos promocionado un BDC mientras el PDC estaba inactivo, cuando vuelva a funcionar encontrará que muchas han cambiado mientras estuvo ausente. Teóricamente debería localizar al nuevo PDC y degradarse automáticamente a BDC. Si no es así, su estado será incierto y se integrará de nuevo en la red indistintamente como estación de trabajo o como PDC inactivo. Al arrancarlo se producirá un error y aparecerá un mensaje indicando que al menos un servicio o unidad a fallado durante la inicialización del sistema. Si miramos el registro del *Visor de sucesos* veremos que ha fallado el servicio NETLOGON y que el fallo se produjo porque ya existe un PDC activo en el dominio.

*Puede ser útil sincronizar la base de datos de directorio antes de promover y degradar servidores*

Para ver cómo se las ha arreglado el antiguo PDC para integrarse en la red nos iremos de nuevo al *Administrador de Servidores* y si aparece como controlador principal de dominio (entonces tendríamos dos controladores principales, el BDC promovido y el antiguo PDC) debemos degradarlo. Para ello seleccionamos el antiguo PDC en la lista de servidores y elegimos la opción *Degradar a controlador de dominio de reserva* del menú *Equipo*. Pese a que en principio aparezca como PDC, no estará activo, ya que al fallar el servicio NETLOGON no participará en la autentificación de usuarios.

Como hemos visto, resulta tremendamente útil sincronizar la base de datos de directorio antes de promover y degradar servidores, si es que tenemos oportunidad para ello. Para forzar una sincronización en el momento que deseemos utilizaremos nuevamente la utilidad *Administrador de Servidores*. Podemos realizar dos tipos de sincronización; a saber, sincronizar un BDC con el PDC o bien sincronizar todo el dominio con el PDC.

Si deseamos sincronizar un BDC, lo seleccionamos en la lista de servidores y en el menú *Equipo* seleccionamos la opción *Sincronizar con el controlador principal de dominio*. Para sincronizar todo el dominio seleccionamos el PDC en la lista y seleccionamos *Sincronizar el dominio completo*.

## Gestión de los ordenadores del dominio

Para acceder a un dominio desde una estación de trabajo basta con tener el protocolo adecuado y conectarse como un usuario del dominio. Sin embargo, con Windows NT Workstation y Server es posible conseguir una relación más estrecha con el dominio, por lo que es conveniente añadirlos explícitamente al dominio. Esto no quita que podamos añadir cualquier otra estación de trabajo al dominio, pero nos ocupará una cuenta de equipo y no conseguiremos nada a cambio. Se pueden añadir ordenadores al dominio de tres maneras diferentes:

- Añadiéndolos durante la instalación. Si lo hacemos así, la instalación deberá llevarla a cabo el administrador del dominio, pues se nos pedirá el nombre y la contraseña del administrador del dominio para poder añadir el nuevo ordenador al dominio.

- Con la Utilidad *Administrador de Servidores*. Para ello seleccionaremos el PDC del dominio y desde el menú *Equipo* elegiremos la opción *Agregar al dominio*, suministraremos el nombre del equipo y pulsamos el botón *Agregar*. Con esto habremos creado una cuenta de equipo; una vez creada la cuenta nos conectaremos al dominio desde ese equipo. Antes de nada comprobaremos que el nombre del equipo y el nombre que suministramos en *Agregar al dominio* coinciden. Si no es así lo cambiaremos y reiniciamos el ordenador. Desde el panel de control pincharemos dos veces el icono *red* y elegimos la solapa *Identificación*, escogemos *cambiar* y tecleamos el nombre del dominio al que pertenece el equipo. Aceptamos y reiniciamos.

- Desde el *Panel de Control*. Pinchamos dos veces en el icono *red* del *Panel de Control* y seleccionamos la solapa identificación. Escogemos *cambiar*, tecleamos el nombre del dominio y marcamos *Crear una nueva cuenta de equipo en el dominio*. Se nos pedirá el nombre y contraseña del administrador del dominio. Aceptamos y reiniciamos.

## La entrada en la red

Ahora sabemos lo suficiente sobre dominios y relaciones de confianza como para comprender perfectamente cómo se validan las entradas a la red.

Los recursos están protegidos a varios niveles y por distintos procesos, pero el acceso general a un dominio o un equipo está protegido por la seguridad de inicio de sesión. Esta seguridad obliga a los usuarios a identificarse en el dominio o en el equipo. El nombre de usuario y la contraseña que escribimos en el cuadro de diálogo *Información de inicio de sesión* se comprueban con la base de datos de directorio del equipo si estamos iniciando una sesión en una cuenta de usuario definida en el equipo o bien se comprobará con la base de datos de directorio del dominio si iniciamos una sesión en una cuenta de usuario de un dominio. La autentificación es lo que hace posible que haya un único inicio de sesión en un do-

minio de Windows NT Server y con él utilizar aplicaciones de la red, o compartir archivos e impresoras.

En un equipo con Windows NT Workstation o en un servidor miembro con Windows NT Server, el servicio *Inicio de sesión* procesa las peticiones de inicio de sesión del equipo local. En un controlador de dominio, el servicio *Inicio de sesión* procesa las peticiones de inicio de sesión en el dominio. El servicio *Inicio de sesión* inicia los procesos: descubrimiento, establecimiento de un canal seguro y autentificación transferida.

El descubrimiento se produce cuando se inicia un equipo con Windows NT Workstation o un servidor miembro con Windows NT Server. El servicio *Inicio de sesión* intentará encontrar un controlador de dominio con Windows NT Server en el dominio en el que se confía. De igual manera, el servicio *Inicio de sesión* de los PDC y los BDC intentará lo mismo con todos los dominios en los que se confía. Una vez descubierto un controlador de dominio, será utilizado para las sucesivas autentificaciones de cuentas de usuario. El canal de comunicación seguro consiste en el envío de mensajes por parte de los servicios *Inicio de sesión* de cada equipo para comprobar la existencia de sus cuentas de equipo. Cuando termina esta comprobación, se establece una sesión de comunicaciones entre los equipos y se utiliza para transferir los datos de identificación de usuario.

La autentificación transferida se produce cuando un usuario inicia una sesión y el equipo que se usa para iniciar la sesión no es un controlador del dominio en el que la cuenta está definida, ni tampoco es el equipo en el que está definida la cuenta de usuario. Entonces el equipo trasladará la información de inicio de sesión, de forma directa o indirecta, a un controlador de dominio en el que esté definida la cuenta de usuario. La autentificación transferida se produce en las siguientes ocasiones:

- Cuando en un inicio de sesión interactivo, el usuario inicia una sesión en un equipo con Windows NT Workstation o Windows NT Server y el nombre indicado en el cuadro Dominio del cuadro de diálogo *Información de inicio de sesión* no es el nombre del equipo. Si sucede esto, el equipo donde se inicia la sesión enviará la petición de inicio de sesión a un controlador de dominio en el dominio al que pertenezca su cuenta de equipo. El controlador primero comprueba el nombre del dominio. Si coincide con el nombre del dominio al que pertenece, el controlador autentifica las credenciales de inicio de sesión comprobándolas con su base de datos de directorio y devuelve la información de identificación de la cuenta al equipo donde se inicia la sesión, permitiendo al usuario conectar con recursos tanto de ese equipo como de todo el dominio. En caso de que el nombre no se corresponda con dominio al que pertenece el controlador de dominio, el controlador de dominio comprueba si se trata de un dominio en el que se confía. Si es así, traslada la petición de inicio de sesión a un controlador del dominio en el que se confía. Ese controlador de dominio autentifica el nombre y la contraseña del usuario con su base de datos de directorio y devuelve la información de identificación de cuenta al primer controlador de dominio, que a su vez la enviará al equipo donde se ha iniciado la sesión. Si el nombre en las credenciales de inicio de sesión no es el nombre del equipo o el nombre del dominio al que pertenece el equipo ni el nombre de un dominio en el que se confía, se considerará que las credenciales presentadas pertenecen a un dominio en el que no se confía, así pues, el inicio de sesión interactivo fallará.

- Si en un inicio de sesión interactivo, iniciamos la sesión en un controlador de dominio pero no en el dominio al cual pertenece el controlador, éste comprobará el nombre del dominio para ver si se trata de un dominio en el que se confía. En caso afirmativo, el controlador transfiere la información de inicio de sesión a un controlador de ese dominio para su autentificación. Si el controlador del dominio en el que se confía autentifica la cuenta, la información de inicio de sesión se transfiere de nuevo al controlador de dominio inicial y el usuario inicia la sesión.

- Durante un inicio de sesión remoto (esto es, establecer la conexión a través de la red). Si el usuario intenta establecer una conexión con un equipo de un dominio en el que no se confía, el inicio de sesión se producirá como si el usuario estuviera conectándose a una cuenta del equipo remoto. El equipo remoto autentificará las credenciales de inicio de sesión con su base de datos de directorio. Si la cuenta no está definida en dicha base de datos, pero en el equipo remoto está activada la cuenta *Invitado* y además no tiene establecida una contraseña, el usuario inicia una sesión con privilegios de invitado. Si no está activada la cuenta *Invitado*, el inicio de sesión fracasa. Si hemos iniciado una sesión en una cuenta de equipo o de dominio y luego intentamos establecer una conexión con otro equipo a través de la red, la autentificación trasferida tiene lugar en forma de inicio de sesión interactivo. Las credenciales utilizadas en el inicio de sesión interactivo se utilizarán para la autentificación transferida, a no ser que especifiquemos otras credenciales escribiendo un nombre de dominio o equipo distintos y un nombre de usuario diferentes en el cuadro *Conectar* (por ejemplo, cuando conectamos unidades de red). Si el equipo al que nos conectamos es un BDC del dominio en el que está definida la cuenta de usuario, pero el BDC no autentifica la contraseña del usuario (por ejemplo, porque la contraseña ha cambiado y el BDC aún no está sincronizado en el momento en que el usuario inicia la sesión), el BDC trasladará la petición de inicio de sesión al PDC del mismo dominio.7

NÚMERO EXTRA

# SÓLO PROGRAMADORES

# LINUX

## Debian 1.3.1

### VERSIÓN OFICIAL en 2 CD-ROM

**2 CD-ROM + MANUAL DE INSTALACIÓN POR SÓLO 2.495 Ptas.**

**MUY PRONTO EN TU QUIOSCO**

### INCLUYE:

- 974 paquetes de Software.
- Miles de páginas de texto en formato HTML.

### CARACTERÍSTICAS PRINCIPALES:

- Han intervenido en esta distribución más de 200. desarrolladores (el mayor grupo Linux).
- Es compatible Slackware y RPM (Red Hat).
- Todo el software incluido ha sido testeado por un excepcional pre-release testing group.

**CON LA GARANTÍA DE UNA DISTRIBUCIÓN OFICIAL**

TOWER

c/ Aragoneses, 7 - 28108 Pol. Ind. Alcobendas (Madrid) - Tel.: (91) 661 42 11* - Fax: (91) 661 43 86
e-mail: solop@towercom.es http://www.towercom.es

WWW

# Netcaster: Iniciación a la Tecnología Push

*Alejandro M. Rayero Abad*
*axl@las.es*

La nueva generación de browsers que incorporan tecnología Push (PointCast, Netscape Netcaster,...) nos introducen en un mundo de "canales" y de "subscripciones", donde el HTML dinámico y JavaScript alcanzan su máxima expresión.

La tecnología Push supone un paso adelante con respecto al HTML. La propuesta de Netscape, Netcaster, está basada en las extensiones HTML (Dinámico y JavaScript) de la propia compañía americana.

Antes de entrar a analizar y programar el Netcaster tendremos que familiarizarnos con una serie de conceptos, hasta ahora desconocidos para el programador de HTML estándar, como el concepto de canal y su diferencia con sitios WEB normales, y los tipos de canales especiales, como el llamado *webtop*.

## El Concepto de Canal

Un canal es una página WEB, orientada a ofrecer información al usuario, que utiliza Netscape Netcaster como medio de transporte hasta el escritorio del usuario. El canal contiene información dinámica, en constante cambio, que llega al usuario de forma transparente, y que puede ser consultado en línea o desconectado. Un canal será pues un conjunto de páginas WEB que utilizan tecnología *PUSH* para enviar su información al usuario de manera automática, sin que haga falta que sea el usuario el que vaya a esa página WEB cada vez que quiere ver nueva información. El usuario puede definir cada cuanto tiempo se actualiza, de manera automática, un canal.

*Un canal es una página WEB que ofrece información al usuario y utiliza Netscape Netcaster como medio de transporte hasta el escritorio del usuario*

Imaginemos un canal con un carrusel deportivo. Un domingo, lo deseable sería que se actualizara cada cinco minutos mientras duren los partidos y que no se actualice durante el resto de la semana. Sin embargo, un canal con noticias sobre informática podría actualizarse una vez al día únicamente.

Los canales Netcaster pueden ser de dos tipos: canales basados en HTML estándar o canales Castanet. Veremos la tecnología Castanet (basado en Java) en futuras entregas de esta serie. Los canales HTML pueden utilizar toda la potencia del browser Netscape Communicator, utilizando HTML estándar, JavaScript, extensiones HTML dinámico, Applets Java, etc.

Las diferencias con una página WEB son obvias, de la necesidad de ver una a una las páginas deseadas, pasamos a recibir, automáticamente, los canales a los que estemos suscritos, los cuales "vendrán" hacia nosotros en intervalos de tiempo que nosotros mismos habremos establecido. De esta manera, el usuario recibe la información actualizada en el momento en el que está disponible, sin que sea necesario tener que ir a buscarla.

Los canales tienen las siguientes características:

- • Los canales son actualizados periódicamente, sin la intervención del usuario.
- • Los contenidos de un canal pueden ser personalizados por el usuario, de manera que éste sólo recibe la información que le interesa.
- • Cierta información de un canal (textos, imágenes, fondos, ...) queda en una caché especial, de manera que no hace falta bajarse el canal entero la próxima vez.

Cualquier página WEB puede convertirse en un canal Netcaster, lo que permite aprovechar las páginas ya existentes, pero las que mejor uso harán de esta tecnología serán las páginas WEB con cantidades de información en constante actualización, como *www.cnn.com* o *www.allgames.com*

*Cualquier página WEB puede convertirse en un canal Netcaster, pero las que mejor uso harán de esta tecnología son las que contienen información dinámica*

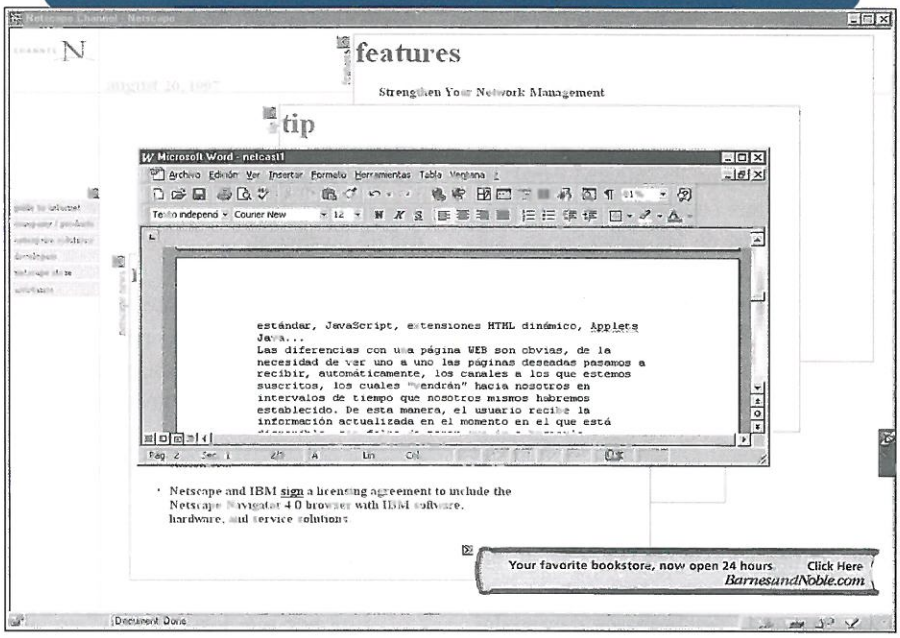
Además, los canales que usen el modo *webtop* dispondrán de características adicionales.

## El Modo Webtop

El modo *webtop* es un tipo de canal que ocupa todo el escritorio, sustituyéndolo (no sólo en la versión windows, Netcaster está disponible, y el tipo de canal *webtop* funciona, en Unix y Mac), de tal manera que el fondo de nuestro monitor se convierte en un enorme canal de información. Un canal *webtop* puede ser definido como una clase de WEB particular, cuyo contenido es visualizado detrás de las ventanas de las aplicaciones del usuario. Ya que los canales *webtop* están siempre presentes, es conveniente que su contenido sea interesante y visualmente atractivo, Netscape también recomienda que los links que contengan los canales *webtop* abran nuevas ventanas en vez de cambiar el contenido del *webtop*.

Vemos en la Figura 1 cómo se trabaja este artículo: Microsoft Word 97 se utiliza como ventana principal y el canal *webtop* llamado "Netscape Channel" como fondo.

Figura 1. Redactor trabajando con un canal web top de fondo.




## Elementos de un Canal

La creación de un canal es, técnicamente hablando, similar a la creación de un WEB, debido a que los canales incluyen elementos HTML, JavaScript y Applets Java. No obstante, tenemos que tener en cuenta las especiales características de un canal.

Un canal está creado sobre la base de ficheros HTML y JavaScript. Hay un fichero HTML principal al que apunta el URL del canal, que es la dirección del mismo. El fichero HTML principal contiene links a otros ficheros que componen el canal.

Otra parte fundamental del canal es una serie de ficheros JavaScript, referenciados mediante TAGs en los ficheros HTML. Estos ficheros JavaScript

confieren capacidades dinámicas al canal. También podemos utilizar ficheros .class (Java) y otros contenidos interpretados por componentes plug-ins instalados en nuestro browser.

## Diseñando un Canal

El diseño de un canal tiene que ir orientado hacia la facilidad de uso, permitiendo que el usuario se pueda mover sin problemas, especialmente en los canales webtop, donde el diseñador renuncia a los botones y menús de un browser estándar.

Con los canales webtop también tendremos que tener en cuenta que los usuarios utilizarán otras aplicaciones que se visualizarán encima de nuestro canal. El canal tendrá que convivir en paz y armonía con el resto de las aplicaciones, sin competir con ellas por la atención del usuario, básicamente, el contenido de un canal webtop ha de ser visible pero sin distraer.

Hemos de tener en cuenta también que el desarrollo de un canal supone la creación de software, estamos creando de cero una interfaz WEB, que no está limitada por una ventana de un browser, fusionando contenido con programa. Netcaster es la plataforma de desarrollo, permitiéndonos utilizar dos intérpretes (HTML y JavaScript) para la presentación y manejo de eventos.

## El Componente Netcaster

Netscape Netcaster está representado en JavaScript mediante el componente *netcaster*, que es un elemento más del array *components*. El array *components* nos informa sobre el estado del componente *netcaster* y, a su vez, las propiedades *name* y *componentVersion* nos dan el nombre y la versión del componente *netcaster*. Otra propiedad, *active*, determina si el componente *netcaster* está activo y funcionando. Finalmente, el método *activate* comienza la ejecución del componente *netcaster*.

El código del canal interactúa con Netcaster a través de un objeto canal, al que accedemos mediante el componente *netcaster*. Cada canal al que estemos suscritos es representado por un objeto canal, teniendo unas propiedades que especifican los parámetros del canal. Para manipular el objeto canal utilizaremos dos métodos que pertenecen al componente *netcaster*: *getChannelObject*, que crea un objeto canal y *addChannel*, que finaliza el proceso suscripción.

Antes de utilizar los métodos del componente *netcaster* tendremos que importarlos en el script en el que los queramos utilizar, debido al modelo de seguridad de JavaScript.

Primero nos aseguraremos de que el componente está activo, para importar sus métodos a continuación, utilizando las siguientes instrucciones JavaScript:

```
import components["netcaster"].getChannel
    Object;

import components["netcaster"].add Channel;
```

Una vez que los métodos han sido importados podremos utilizarlos localmente sin necesidad de hacer referencia explícitamente al método *netcaster*.

## Proceso de Suscripción

Este proceso se inicia cuando un usuario añade un canal a Netcaster. Para el programador comienza en el momento en que un script obtiene el objeto canal utilizando el método *getChannelObject* y finaliza en el momento en que se llama al método *addChannel*.

## *El proceso de suscripción se inicia cuando un usuario añade un canal a Netcaster*

El WEB desde el que iniciamos el proceso de suscripción tendrá que incluir un botón o link que invoque las llamadas al API *getChannelObject* y *addChannel*. Más adelante veremos algunas de las propiedades Netcaster en detalle, pasando después a analizar un ejemplo de creación de un canal. Para ver más ejemplos y un "wizard" que creará automáticamente el código JavaScript necesario para el proceso de suscripción a un canal, véase la página WEB:

*http://developer.netscape.com/library/examples/index.html*

Cuando hacemos la llamada a *addChannel*, Netcaster utiliza los valores de las propiedades del objeto canal como los valores por defecto, presentando una caja de diálogo al usuario para su confirmación.

El usuario puede cambiar cualquiera de los valores de las propiedades a través de la caja de diálogo llamada "Channel Properties", que se invoca tras pulsar el botón "Properties" en la caja de diálogo de confirmación. Los cambios hechos por el usuario sustituyen a las propiedades especificadas en el objeto canal, además, no hay valor de retorno para conocer las modificaciones que ha hecho el usuario.

El WEB en el que nos suscribimos al canal puede tener opciones que ayuden al usuario a configurarlo, pero el último paso lo da el propio usuario mediante la caja de dialogo de confirmación, la cual es el ultimo y necesario paso para la suscripción del canal.

# Propiedades Netcaster

El array de componentes tiene tres propiedades que contienen información sobre el componente *netcaster*: *active, componentVersion* y *name.*

- **active:** Es una propiedad, devuelve *true* si el componente está instalado y funcionando. Su sintaxis es: *components["netcaster"].active*

- **componentVersion:** Es una propiedad que contiene el número de versión del componente. Su sintaxis es: *components["netcaster"].componentVersion,* devolviendo el valor 1.0 para la versión de Netcaster contenida en Communicator 4.02.

- **name:** Propiedad que devuelve el nombre del componente (Netcaster en este caso). Su sintaxis es: *components["netcaster"].name*

# Métodos Netcaster

El componente *netcaster* tiene tres métodos: *activate, getChannelObject* y *addChannel.*

- **activate:** Es el método que inicia la ejecución del componente. Su sintaxis es: *components["netcaster"].activate();*

- **getChannelObject:** Devuelve un nuevo objeto canal preinicializado con valores por defecto. Su sintaxis es: *components["netcaster"].getChannelObject();*

- **addChannel:** Añade un canal a Netcaster. Su sintaxis es: *components["netcaster"].addChannel(channelObject);* siendo *channelObject* una variable que representa el objeto canal anteriormente devuelto por el método *getChannelObject.*

# Propiedades del Objeto Canal

Los objetos canal tienen una serie de propiedades que son utilizadas para especificar los parámetros del canal. Como hemos visto anteriormente, primero llamamos al método *getChannelObject,* que nos devuelve un nuevo objeto canal, a continuación asignamos valores a las propiedades del objeto canal (por defecto estas propiedades ya tienen un valor) y finalmente llamamos al método *addChannel* para completar el proceso de suscripción.

*Los objetos canal tienen una serie de propiedades que son utilizadas para especificar los parámetros del canal*

- **intervalTime:** Es la propiedad que indica el intervalo de tiempo que pasa entre que el usuario se baja en contenido del canal. Su sintaxis es:
*channelObject.intervalTime = periodo.*
El valor *periodo* es un entero positivo que representa ese intervalo en minutos. También puede tomar uno de los valores especiales negativos: -2 indica cada 5 minutos, -3 cada media hora, -4 cada hora, -5 a diario y -6 semanalmente.

- **absoluteTime:** Es la propiedad que especifica el horario en el que el usuario se bajará el contenido del canal. Su sintaxis es: *channelObject.absoluteTime = horario.*

El valor *horario* es un entero que representa el tiempo absoluto, en minutos después de la medianoche del domingo, en los que el canal se enviará al usuario. Esta propiedad es necesaria si la propiedad *intervalTime* tiene un valor de -5 o -6, si tiene otro valor, *absoluteTime* es ignorado.

- **depth:** Especifica la jerarquía de links del canal a ser bajados en el caché. Su sintaxis es:
*channelObject.depth = valor*

- **desc:** Es una cadena con el texto descriptivo del canal. Su sintaxis es:
*channelObject.desc = "cadena"*

- **estCacheSize:** Especifica la caché estimada que necesitará el canal (en bytes). Su sintaxis es:
*channelObject.estCacheSize = tamaño.*
Este valor será presentado al usuario, que tendrá que comprobar si hay suficiente espacio en su disco duro antes de suscribirse al canal (o no suscribirse).

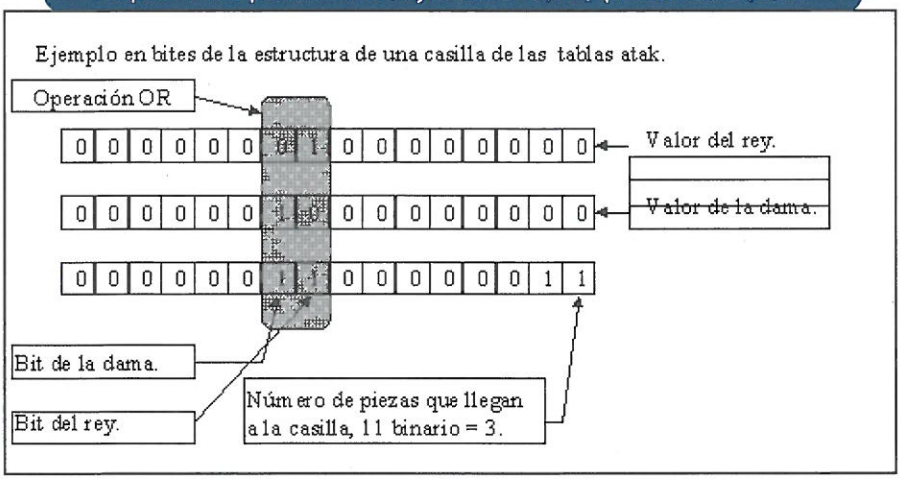
- **heightHint:** Especifica la altura de la ventana de nuestro canal (en pixeles). Su sintaxis es:
*channelObject.heightHint = valor*

- **leftHint:** Sitúa la ventana del canal, relativamente en píxeles, al límite izquierdo de la pantalla. Su sintaxis es:
*channelObject.leftHint = valor*

- **maxCacheSize:** Especifica el tamaño máximo del caché del canal (en bytes). Su sintaxis es: *channelObject.maxCacheSize = tamaño*

- **mode:** Especifica el modo de presentación del canal. Su sintaxis es: *channelObject.mode = "string"*, donde *string* puede tomar tres valores: *window, full* y *webtop. Window* abre

Figura 2. Pretendemos establecer una tarifa de 250 pesetas por hora de conexión. Así pues el multiplicador será 250 y el divisor 60, o lo que es lo mismo, 25/6.




una ventana navigator, *full* abre una ventana sin menús que ocupa toda la pantalla, pero sin sustituir al escritorio, cosa que ocurriría si escogemos el modo *webtop*.

• **name:** Le asigna un nombre al canal. Su sintaxis es:
*channelObject.name = "nombre"*

• **topHint:** Especifica la posición vertical deseada con respecto al límite superior del escritorio (en píxeles). Su sintaxis es:
*channelObject.topHint = valor*

• **url:** Especifica la dirección internet (URL) del canal. Su sintaxis es:
*channelObject.url = "cadena"*

• **widthHint:** Especifica la anchura deseada de la ventana (en píxeles). Su sintaxis es:
channelObject.widthHint = valor

## Comportamiento de la Caché

Como habíamos prometido, a continuación vamos a analizar un ejemplo de código (añade.htm) JavaScript para la creación de un objeto canal, añadiéndolo a Netcaster.

Cuando Netcaster está suscrito a un canal, automáticamente conecta con el URL del canal en los intervalos especificados en las propiedades del canal. Cuando la conexión está activa, Netcaster se baja el contenido del canal mediante un proceso conocido como "Crawling". El proceso de "Crawling" comienza por la página principal del canal y continúa por todas las páginas vinculadas, hasta que llega a la profundidad especificada en las propiedades del objeto canal.

Todos estos contenidos son almacenados en una caché especial de Netcaster.

*Crawling es el proceso por el cual Netcaster se baja el contenido del canal teniendo en cuenta las propiedades del objeto*

Cuando el "Crawler" de Netcaster conecta con una página se baja primero el fichero HTML y todos los ficheros HTML hasta llegar a la profundidad indicada, volviendo después atrás para continuar por las imágenes contenidas en esas páginas, de arriba a abajo.

### Añade.htm

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript1.2">

function addMyChannel(name,url) {
   var nc = components["netcaster"];
   nc.activate();
   if(nc.active == true) {
      import nc.getChannelObject;
      import nc.addChannel;
      channel = getChannelObject();
      channel.url=(url ||
                  "http://xvt.allgames.com");
    channel.name = (name ||
                  "X-Wing vs. TIE-Fighter.");
      channel.desc = name;
      channel.intervalTime = -5;
      channel.absoluteTime = 0;
      channel.estCacheSize = -1;
      channel.maxCacheSize = 500000;
      channel.depth = 3;
      channel.active = 1;
      channel.topHint = screen.availTop;
      channel.leftHint = screen.availLeft;
      channel.widthHint =  640;
      channel.heightHint = 480;
      channel.mode="webtop";
      addChannel(channel);
   } else { //Si Netcaster no esta activo:
      loadWin = window.open('javascript:
                  document.open();'+
                  'document.write("Cargando
Netcaster...");
        document.close();',
        'load','width=200,height=100');
        setTimeout('if(loadWin.close)
        loadWin.close();',5000);
   }
}

</SCRIPT>
</HEAD>

<BODY>

<FORM>
<INPUT TYPE="button" VALUE="Añade el
Canal" onCLick="addMyChannel();">
</FORM>

</BODY>
```

Finalmente se baja el resto de los ficheros contenidos en las páginas, como Applets Java o ficheros JavaScript (.js). De esta manera el usuario dispondrá en primer lugar del texto en caso de que la comunicación se corte.

En el caso de que algún objeto de las páginas no pudiera ser traído en la actual conexión (debido a un corte de la misma, un link mal puesto en la página, etc.) Netcaster trataría de traerlo la siguiente ocasión que se encuentrara conectado.

Los contenidos que sean enviados en directo (audio, vídeo, etc.) no podrán, sin embargo, quedarse en la caché.

## Un Robot en mi Página

Un "crawler" es un Robot de búsqueda de información a través de páginas WEB (ya sea el contenido en Netcaster o los de los motores de búsqueda). Netcaster permite que controlemos su robot, de tal forma que algunas de las páginas no sean bajadas a la caché del usuario por el robot, siendo esos links ignorados.

El fichero que controlará a los robots tendrá que llamarse robots.txt, y además deberá estar contenido en el mismo subdirectorio que la página principal de nuestro canal. El "crawler" intentará obtener el fichero robots.txt en primer lugar, analizándolo y siguiendo las instrucciones que en él le indicamos. Si el fichero no existe, entonces el robot entiende que no hay instrucciones específicas para él, y comenzará a bajarse todos los contenidos de nuestra página WEB.

El fichero robots.txt contendrá una serie de registros, separados por líneas en blanco.

Un registro es una serie de líneas, que contendrán el carácter dos puntos (":"). Los registros comienzan por una o más líneas con la palabra *User-agent*. Estas líneas especifican el robot al que aplicarán el registro. Un asterisco ("*") supone que la información va dirigida a todo tipo de robots.

A continuación del *User-agent* encontramos una serie de líneas que indican el URL de un directorio o de un fichero individual que ha de ser ignorado y no bajado. Estas líneas han de comenzar con la palabra *Disallow*, seguida de dos puntos y el directorio o el nombre del fichero.

*El fichero robots.txt se encargará de controlar los robots de búsqueda de información, incluido el de Netcaster*

El fichero robots.txt puede contener comentarios y líneas que comienzan con el carácter almohadilla ("#") que son ignoradas.

Un ejemplo de robots.txt podría ser el siguiente:

```
# robots.txt de ejemplo
User-agent: *

Disallow: /ficheros/zips

Disallow: /secreto/encuentralo.html
```

Netcaster se baja automáticamente un canal una vez que ha sido suscrito. Para forzar a Netcaster a bajarse un canal basta con pulsar el botón *options*, escoger un canal y pulsar el botón *Update Now*. Cuando la barra roja y verde que está situada debajo del canal se detiene, entonces el canal ya está disponible en nuestro disco duro.

Si la barra se queda en color verde, esto significa que el canal ha llegado completo y sin problemas; si el color es rojo entonces nos está indicando que ha ocurrido algún error.

## Conclusiones

En este artículo hemos aprendido a manejar el Netscape Netcaster, hemos conocido conceptos nuevos como "Push", "Crawler" y "Suscripción" y ya sabemos cómo crear nuestro propio canal de información "Push".

En futuros artículos aprenderemos a dar vida al canal mediante JavaScript y HTML Dinámico, diferenciando las necesidades de un canal con respecto a una página WEB.

## Bibliografía


- "HTML Dinámico con Netscape Communicator y JavaScript (Hojas de Estilo)", por Alejandro M. Reyero. Sólo Programadores número 35.
- "HTML Dinámico con Netscape Communicator y JavaScript II (Layers)", por Alejandro M. Reyero. Sólo Programadores número 36.


## Contactar con el autor

Si el lector quiere hacer llegar sus opiniones sobre los artículos de la serie sobre WWW, preguntas, ejemplos, o, simplemente, decir "hola" al autor, puede hacerlo enviando un mail a través de la dirección de correo electrónico siguiente:

E-Mail: axl@las.es
WEB: http://xvt.allgames.com

CRIPTOGRAFÍA

# Autentificación. Kerberos

*José Antonio Mendoza*

En los sistemas informáticos, tan importante como la transmisión segura de los datos es controlar el acceso de personas y programas a los servicios prestados por éstos. Sobre todo en entornos abiertos, es decir, aquellos en los que cualquier persona puede ponerse delante de un ordenador del sistema, e intentar abrir una sesión.

En los anteriores artículos de esta serie hemos visto algoritmos mediante los cuales se cifraba y firmaba la información a transmitir a través de un canal potencialmente inseguro, y la manera de protegernos de los diferentes y múltiples tipos de ataques, a cual de ellos más pintoresco. De vez en cuando salta a los medios de comunicación la desarticulación de una banda de "hackers" que ha sido capaz de conectarse a ordenadores a los que no tenían acceso y realizar todo tipo de fechorías, desde dejar simple constancia de haber estado allí hasta provocar pérdidas irreparables en la confidencialidad de datos estratégicos, pasando por dejar fuera del control de los operadores a los sistemas conquistados. Veamos ahora cómo protegernos del acceso de personas y programas no autorizados a los recursos informáticos disponibles en una red.

En la vida real las personas autentificamos por teléfono a nuestro interlocutor gracias a la voz. El ancho de banda de un circuito telefónico viene determinado por el mínimo necesario como para reconocer la voz de la otra persona. Al comienzo de la telefonía el ancho de banda era un recurso muy escaso, es decir caro, por lo que había que encontrar ese límite. Por eso por teléfono reconocemos al interlocutor pero no oímos su voz completa. La voz humana comprende un rango de frecuencias que va de los veinte hercios a los veinte mil, sin embargo cuando oímos por teléfono lo hacemos entre los trescientos hercios y los tres mil cuatrocientos. Esa es la razón de que por teléfono no oigamos la voz completa. Cuando las personas se reúnen sin conocerse de antemano, van a un centro de acreditación donde a uno le dan, tras presentar un documento de identidad, una tarjeta, con su foto, su nombre, su procedencia etc. y una pinza para prenderla en la solapa, permitiéndole acceder a unos determinados recintos mientras que a otros el acceso permanece prohibido. Los ordenadores necesitan programas y protocolos capaces de realizar la tarea que los humanos realizamos con la voz, con los centros de acreditación, con las tarjetas etc. Cuando a un sistema informático llega la petición de un servicio, dicho sistema debe responder a dos cuestiones. La primera, ¿es un impostor, o es el que dice ser el demandante del servicio?, la segunda, ¿dicho demandante tiene derecho al servicio que solicita?. Una vez autentificado el interlocutor, comprobar si tiene o no acceso a un determinado servicio, se reduce a consultar una tabla. Estas cuestiones no son fáciles de resolver, y como siempre, algo que se supone seguro lo es hasta que se demuestra lo contrario.

## Intercambio de contraseñas Diffie-Hellman

Cuando dos desconocidos quieren entablar una comunicación segura y necesitan para ello, compartir una contraseña, tienen que

encontrar un medio de poderse enviar dicha información, sin que caiga en manos extrañas. Una primera aproximación al problema la dieron Diffie y Hellman en 1976, estableciendo el siguiente protocolo.

Pepe quiere intercambiar de forma segura con Juan una contraseña. Lo primero que hace Pepe es enviar un mensaje a Juan con los siguientes datos ( n, g, $g^x$ mod n ), Juan responde a Pepe enviándole ($g^y$ mod n ). La contraseña compartida por los dos sería K= $g^{xy}$ mod n. Para que el algoritmo sea seguro debe cumplirse que n y g sean dos números primos elevados, que (n-1)/2 también sea primo y que g sea menor que n, y por último que n sea raíz primitiva de g. Esto último quiere decir que todas las potencias de n hasta la g-1 a las que se les aplica la operación módulo n dan un conjunto de g-1 números todos ellos distintos.

*El protocolo de intercambio de claves Diffie- Hellman es una muestra de la aplicación de las propiedades de los números primos*

Todo lo anterior es posible debido a que se cumple una propiedad de la aritmética modular: ( $g^y$ mod n )$^x$) mod n = ( $g^x$ mod n )$^y$) mod n = $g^{xy}$ mod n. Por lo tanto Juan elevará ( $g^x$ mod n ) a "y", para calcular K y Pepe al recibir ( $g^y$ mod n ) lo elevará a x, ambos, lógicamente, a sus respectivos resultados le aplican la operación módulo n.

Un adversario que interceptara los mensajes se haría con las dos potencias pero no dispondría ni de x ni de y, que son necesarias para calcular K. El impostor puede pensar en calcular el logaritmo discreto de la potencia ya que conoce la base, "g", pero resulta computacionalmente imposible calcular el logaritmo discreto cuando los valores que intervienen son tan elevados como en este caso.

Dos usuarios que no se conocían comparten ahora una contraseña segura. Como no podía ser menos se encontró una debilidad, el ataque "bucket brigade" o ataque de la cadena de cubos. Este nombre tiene su origen cuando, al producirse un incendio se crea una cadena de personas que llevan agua desde la cuba de agua hasta el fuego, mediante cubos que van pasando de mano en mano. ¿Qué tiene que ver el incendio con el ataque a nuestro protocolo?, pues el adversario Pedro intercepta la comunicación entre Pepe y Juan haciendo llegar a Juan el mensaje de Pepe adulterado: en vez de ( n, g, $g^x$ mod n ) manda a Juan ( n, g, $g^z$ mod n ). Juan contesta a Pepe con ( n, g, $g^y$ mod n ) que Pedro también intercepta, por último Pedro manda a Pepe ($g^z$ mod n ). Es como si Pedro se interpusiera en la cadena de cubos entre Pepe y Juan y cambiara el agua por gasolina en sentido del fuego y llenara el cubo de piedras en sentido la cuba. Esto es posible debido a que Pepe y Juan no se conocen y no ven el contenido de los cubos. Pedro hace una doble suplantación.

¿Cómo pueden defenderse Pepe y Juan de Pedro que les intercepta los mensajes y se los manipula ? Recordemos que el medio es público y debemos suponer que dichos ataques son posibles. Otra vez aparecen dando una solución Rivest y Shamir. En este caso propusieron el *interlock protocol*, que consiste en un truco muy simple: enviar sólo parte del mensaje, por ejemplo los bits pares, y una vez enviados éstos enviar los impares posteriormente. Pero eso sí, esta vez haciendo uso de clave pública y privada.

Veamos más de cerca cómo se evita que Pedro nos juegue una mala pasada. Supongamos que Pepe y Juan poseen la clave pública de su compañero y que es la verdadera, no la de un impostor. Pepe encripta el mensaje para Juan pero sólo le manda los bits pares, cuando éste los recibe le manda a Pepe los bits pares de su mensaje cifrado. Pepe al recibirlos manda ahora los impares, por lo que Juan descifra el mensaje haciendo uso de su clave privada y manda sus bits impares a Pepe realizando este la misma operación.

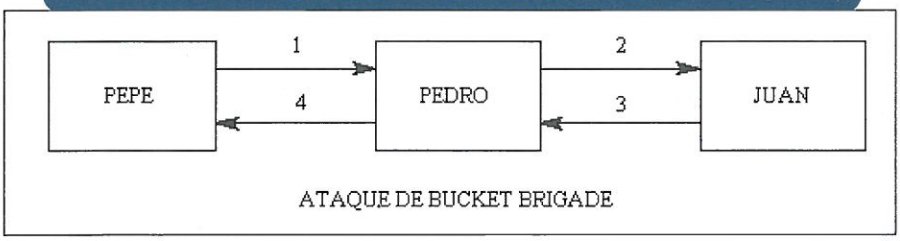
Supongamos ahora que Pedro mediante el ataque *bucket brigade* ha sido capaz de manipular el envío de las claves públicas de Pepe y Juan de tal manera que la clave pública que Pepe le envía a Juan, Pedro la intercepta y en su lugar manda la suya propia, haciendo lo mismo con la que envía Juan a Pepe. De esta manera Pedro está en condiciones de suplantar a los dos interlocutores enterándose y, si le interesa, manipulando la conversación.

Analicemos pues cómo afecta a Pedro el envío de los mensajes de manera incompleta. Pepe cifra su mensaje con la falsa clave pública de Juan que en realidad es la de Pedro, pero sólo manda los bits pares del mensaje cifrado. Cuando Pedro intercepta dicho mensaje no lo puede descifrar con su clave privada al no estar el mensaje entero, si se pusiera a descifrarlo solo obtendría basura. Necesita los bits impares, pero éstos Pepe no los mandará hasta que haya recibido algo de Juan. Juan no manda nada hasta que reciba algo de Pepe. Recordemos que ahora el mensaje se encuentra a mitad de camino en las manos de Pedro.

Pedro no puede mandarle a Juan los bits pares pues han sido cifrados con una clave pública falsa, para hacerle llegar el mensaje es necesario que los descifre con su clave privada y los cifre con la verdadera clave pública de Juan, pero no lo puede hacer al estar el mensaje incompleto. En este momento Pedro sólo puede inventarse un mensaje y mandar unos bits pares inventados cifrados con la clave pública de Juan. El proceso se repetiría y Juan y Pepe terminarían dándose cuenta de que alguien está manipulando sus mensajes.

En vista de la posibilidad de que alguien suplante a un interlocutor, los diseñadores de protocolos, pensaron en la posibilidad de la existencia de un tercero, que se encargue de generar y entregar, de forma segura, claves de sesión a

Figura 1. Secuencia de mensajes.




dos usuarios que estén interesados en mantener un conversación. A ese tercero se le llama centro de distribución de claves ( KDC ), lo llamaremos así en adelante.

## Centro de distribución de claves

Uno de los inconvenientes del intercambio de claves entre interlocutores es que, cuando el número de estos se hace grande, resulta ingobernable el procedimiento. Terminamos por no controlarlas, las claves se cambian con cierta periodicidad, se producen altas, bajas; en resumen, debemos preocuparnos de mantener una base de datos que gestione dicha problemática.

A su vez, como hemos visto anteriormente, el intercambio de claves no es un asunto trivial, los impostores siempre pueden estar al acecho. Por lo tanto si delegamos dichas operaciones en un centro seguro en el que podamos confiar que pueda actuar de notario, dando fe de que los usuarios son los que son, entonces nos quitaríamos un gran peso de encima.

Empezaremos por los protocolos más sencillos e intentaremos llegar hasta aquellos que se están empleando en la actualidad, como podría ser el protocolo Kerberos que, además de autentificar, tiene algunas otras funciones que pasaremos a comentar más adelante.

*El Centro de distribución de claves se hace necesario cuando el número de interlocutores es muy elevado*

La situación ahora sería la siguiente: tenemos un centro de distribución de claves que tiene en su base de datos una clave por usuario y que sólo ellos dos conocen. Cuando Pepe quiere entablar conversación con Juan, únicamente tiene que mandar al KDC un mensaje compuesto por su identidad y por otra parte cifrada con $K_J$,( clave de Juan conocida por él y el KDC ), conteniendo la futura clave de sesión a usar con Pepe y la identidad del usuario con el que Juan quiere entablar conversación, en este caso Pepe.

El KDC reconoce la procedencia del mensaje y descifra la parte cifrada con la clave de Juan, viendo que quiere entablar conversación con Pepe y guardando la clave de sesión propuesta. Ahora el KDC manda a Pepe un mensaje cifrado con la clave de Pepe conteniendo la identidad del usuario que quiere hablar con él y de la clave de sesión a utilizar. Pepe y Juan ya pueden hablar cifrando los mensajes con Ks (clave de sesión). Hay que recordar que las claves que comparte el KDC con cada uno de los usuarios son auténticas: no ha habido ni impostores ni manipuladores, por eso cuando se descifra un mensaje y se comprueba que tiene sentido y si no aparece basura es que podemos estar seguros que el mensaje tiene una procedencia legítima.

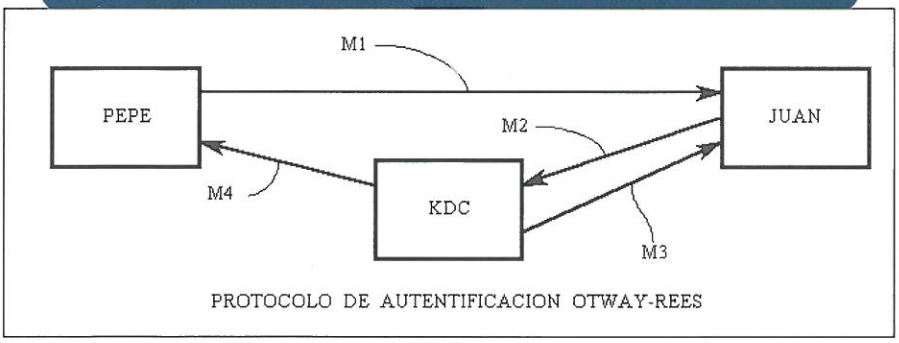
Seguro que ya lo estabamos esperando ¿el protocolo anterior tiene un agujero de seguridad? Pues sí, lo tiene: el ataque ahora se llama el de la repetición "replay attack". Pedro, como siempre escuchando la red, procede de la siguiente manera: Graba un mensaje de Pepe a Juan y Pedro se pone a retransmitirlo. Cuando los mensajes le llegan a Juan, está seguro de que vienen de Pepe pues después de descifrarlo con la clave de sesión es un mensaje con sentido aunque idéntico a uno de los recibidos durante la misma. Si la operación es hacer un ingreso de dinero a un tercero a éste le ha tocado la lotería, o lo mas probable es que esté asociado a nuestro hábil Pedro.

Para solucionar el problema se recurre a conseguir que los mensajes tengan alguna información que los haga únicos, como el tiempo de emisión, o números aleatorios, de tal manera que cuando lleguen dos o más mensajes iguales se descarten todos los posteriores al primero en llegar.

Para solucionar la debilidad anterior Otway y Rees propusieron en 1987 un protocolo que resumidamente proponía lo siguiente. Suponemos que Pepe quiere hablar con Juan, empieza mandándole a Juan el mensaje [P,J,A,$K_P$( P,J,A, $R_P$)], donde P es un identificador de Pepe, J un identificador de Juan, A un número aleatorio, $K_P$ la clave que Pepe comparte con el KDC, con la que se cifra (P,J,A,$R_P$), el resto del mensaje, $A_P$ es un segundo número aleatorio necesario para evitarse réplicas en posteriores mensajes. Seguidamente Juan, que sólo entiende de este mensaje la parte no cifrada. manda al KDC [ P, $K_P$( P,J,A, $R_P$), J,$K_J$( P,J,A,$R_J$)], cuando el KDC recibe este mensaje manda uno a Pepe $K_P$($R_P$,Ks ) y a Juan $K_J$($R_J$,Ks ). Pepe y Juan disponen de una clave de sesión Ks con la que enviar sus mensajes cifrados a la otra parte.

Veamos más de cerca la función de cada componente de los mensajes. P y J identifican a los interlocutores, A es un número aleatorio usado como identificador común, en caso de que el KDC no los encontrara al descifrar las dos partes cifradas

Figura 2




PROTOCOLO DE AUTENTIFICACION OTWAY-REES

del segundo mensaje sería porque alguien ha adulterado el contenido de alguno de ellos. $K_P$ y $K_J$ son las claves que identifican a Pepe y Juan ante el KDC. Por último $R_P$ y $R_J$ les sirven a Pepe y Juan para asegurarse de que el mensaje que les llega con Ks, no es una copia de otro anterior, que casan con el número aleatorio contenido en la parte cifrada del primer mensaje.

## Kerberos

Mediante los anteriores protocolos nos hemos familiarizado con la problemática relacionada con el intercambio de mensajes y claves y, en resumen, con la forma de mantener una conversación a salvo de los peligros que nos rodean en una red donde el acceso es libre. De nada nos sirve el más sofisticado método de criptografía si no podemos intercambiar de una manera segura las claves, o si nos graban y repiten conversaciones, entre otras cosas.

Hasta ahora sólo nos hemos preocupado de la autenticidad de los mensajes y los interlocutores, pero nos hemos olvidado del acceso a los recursos. Cuando nos conectamos a una máquina para demandar un determinado servicio, primero ésta se asegura de que somos en realidad quien decimos ser y posteriormente comprueba si el servicio que requerimos entra dentro de los que tenemos acceso.

A todos estos desafíos da solución Kerberos, un protocolo desarrollado en el M.I.T. como parte del proyecto Atenea, en el que tomaron parte las siguientes personas: Steve Miller y Clifford Neuman conjuntamente con Jerome Salzter y Jeffrey Schiller, todos ellos personal del MIT, los dos primeros como miembros del staff de dicho proyecto, el tercero director técnico del proyecto y el último Network Manager del MIT.

Cerbero es el perro de tres cabezas que en la mitología Griega guardaba la puerta del infierno. La palabra cancerbero usada con asiduidad por los locutores deportivos como sinónimo de portero proviene de este nombre. Este protocolo se encuentra explicado, incluido en pseudocódigo en la RFC-1510, para los lectores que quieran profundizar más en él.

En este artículo vamos a ver la versión cuatro como base del estudio y mencionaremos aquellas partes de la versión cinco que han sido mejoradas.

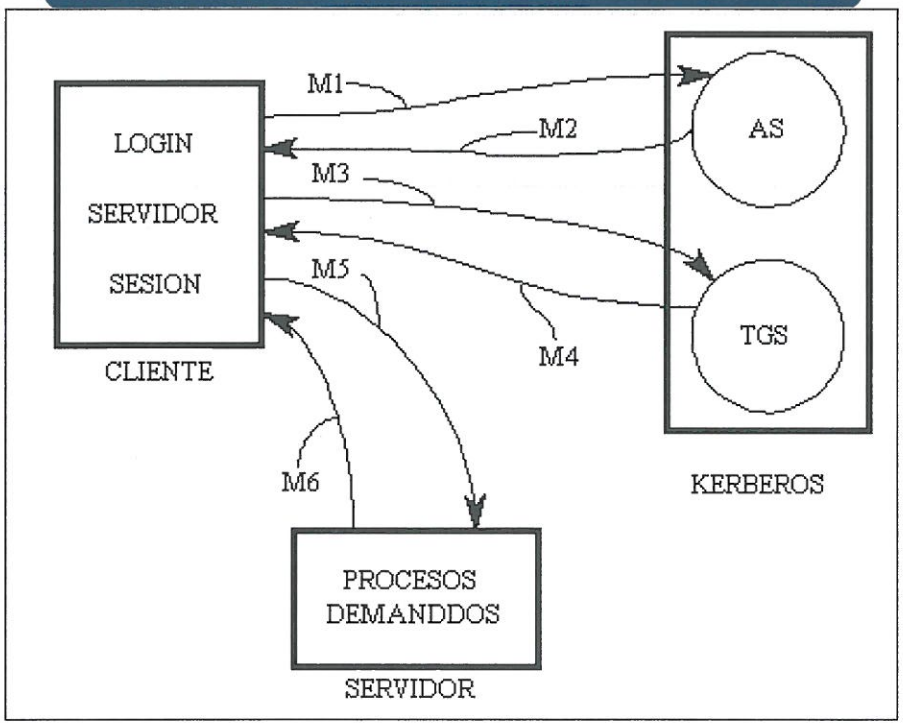
Antes de describir cómo funciona y qué prestaciones nos brinda Kerberos, tenemos que mencionar las condiciones que debe cumplir la red en la que lo haremos trabajar, para que lo haga de una manera segura. Son las siguientes :

1.- Kerberos no soluciona el problema de la seguridad de forma integral, los ataques pueden ir dirigidos a los sistemas operativos o a otras partes de la red. En estos casos son los administradores los encargados de velar por la seguridad.
2.- Los administradores deben mantener sus claves seguras, cualquier filtración resulta fatal para la seguridad de la red, mascaradas y suplantaciones pueden ser entonces posibles.
3.- Los usuarios deben elegir sus claves de forma robusta, deben incluir alguna mayúscula, algún número, un carácter tipo guión, igual, etc. De esta manera hacemos imposibles los ataques a base de diccionarios. Arcadia, romeo, kerberos, no son claves adecuadas. Claves del tipo "5RosaS;)" son más recomendables.
4.- Kerberos hace uso de "Timestamp", o sellos temporales, es decir en algunos mensajes se añade la fecha y la hora aparece con una determinada precisión. Por tanto los relojes de las máquinas del sistema deben estar sincronizados. La sincronización de los relojes de una red no es un problema trivial, muy al contrario, es un asunto muy complejo. Un ordenador que actúa de reloj patrón, suministra la hora a los demás de la organización, pero los mensajes que portan dicha información, tardan en llegar a su destino un tiempo a priori indeterminado, por lo que la hora contenida en el mensaje al llegar a su destino ya no es la hora actual. Por eso los relojes de los ordenadores de una red no marcan la misma hora, ésta se encuentra dentro de un rango mas o menos controlado.
5.- Los identificadores de los administradores deben tratarse con sumo cuidado, sobre todo a la hora de realiizar altas y bajas, para que no haya malentendidos. No es conveniente reciclar los nombres.

Hemos visto lo que kerberos exige a sus usuarios, veamos ahora los servicios que kerberos va a prestar :

1.- Seguro. Un husmeador, persona que escucha la red, grabando todo el tráfico durante un intervalo de tiempo en un disco duro, no va a ser capaz de suplantar a un usuario legal.
2.- Fiable. Los servicios que se basan en Kerberos no van a verse alterados por la caída de un servidor. La

Figura 3. Protocolo Kerberos.




arquitectura del protocolo es de tal manera que unos se respaldan a otros.

3.- Transparente. Al usuario no le interesa cómo funciona el protocolo por dentro, sólo le interesa que sus servicios sean eficientemente satisfechos, molestándosele lo mínimo posible. Un sistema que cada vez que tecleas un comando te pide la "password" sería insufrible.

4.- Escalable. Con el ritmo frenético que están tomando las cosas, y la rapidez con que crecen los sistemas, Kerberos debe estar abierto a la posibilidad de que la red crezca, sin que por ello deje de poder usarse.

## Versión 4

Kerberos involucra a las siguientes entidades: la estación de trabajo donde se conecta el usuario, la máquina donde está kerberos esperando peticiones y el server del que el usuario pretende unos determinados servicios. Kerberos a su vez consta de dos servicios: el servidor de autentificaciones y otro de tickets de garantía.

Para describir mejor el protocolo a cada entidad le vamos a dar un acrónimo . Así al cliente le llamaremos C, al "Authenticaction Server" AS, al servidor de tickets o " Ticket Granting Server " TGS y al Servidor de servicios S.

Su uso comenzaría de la siguiente forma: un usuario cualquiera se pone delante de un ordenador de la red, teclea su *username*, y a continuación el ordenador le pregunta su password. El programa cliente manda el mensaje uno al AS y cuando recibe el mensaje dos, comprueba que se pueda descifrar con la clave derivada de la password tecleada, generalmente una función hash. Si obtiene un mensaje con sentido ya dispone del ticket que lo autentifica.

Una vez que tenemos el ticket de autentificación, podemos disponernos a conseguir el segundo. Mandamos el mensaje tres, cuando éste llega al TGS lo analiza y comprueba si dicho usuario tiene derecho al servicio demandado, en caso afirmativo manda el ticket de garantía al cliente con el mensaje cuatro. Solo nos queda hacer uso del nuevo ticket con el servidor, el cual nos prestará el servicio demandado, mensajes cinco y seis.

*Kerberos pretende ser un protocolo que sea seguro, fiable, transparente y escalable*

Cada vez que requiramos un servicio, el sistema, automáticamente y sin consultarnos, se encarga de gestionar la demanda del segundo ticket, sin tener que volver a autentificarnos como al principio.

El protocolo está compuesto de seis mensajes. Por orden en el tiempo serían dos entre C y AS, dos entre C y TGS y dos entre C y S. El contenido de los mismos lo pondremos entre llaves. Son los siguientes:

• **Mensaje 1 : de C a AS : { $ID_C$, $ID_{TGS}$, $TS_1$ }.**

EL cliente demanda el ticket de garantía.

$ID_C$ : Le comunica a AS la identidad del cliente.

$ID_{TGS}$ : Le comunica a AS que el usuario quiere comunicarse con TGS.

$TS_1$ : Le permite verificar al AS, que el reloj del cliente está sincronizado con el suyo propio.

Este primer mensaje es bastante simple. Sólo reseñar la necesidad, como dijimos al principio, de que los relojes del sistema se encuentren sincronizados. Esto garantiza la "frescura de los mensajes".

• **Mensaje 2: de AS a C: { $Ekc[ K_{C,TGS}$, $ID_{TGS}$, $TS_2$, $Lifetime_2$, $Ticket_{TGS}$,] }.**

El AS envía el ticket pedido.

Ekc : Contraseña del cliente que se usa para cifrar el mensaje y para verificarse. Hay que recordar que cliente y AS mantienen una clave que solo ellos dos conocen.

$K_{c,TGS}$ : Clave de sesión con la que cliente y TGS mantendrán la privacidad de los mensajes.

$ID_{TGS}$ : Gracias a la inclusión de la identidad del TGS el AS confirma que el ticket es para usarlo con el TGS.

$TS_2$ : Le dice al cliente la hora en la que el ticket fue generado.

$Lifetime_2$ : los tickets no duran eternamente, con este valor se le comunica al cliente durante cuánto tiempo el ticket entregado tiene valor, es una fecha de caducidad.

$Ticket_{TGS} = Ek_{TGS}[K_{c,TGS}, ID_c, AD_c, ID_{TGS}, TS_2, Lifetime_2]$. Este campo exime al usuario de tener que teclear una y otra vez la clave cada vez que cambie de servicio requerido.

Vemos cómo este mensaje se complica un poco más. Gracias a ir cifrado se garantiza la privacidad de la información contenida y a su vez el cliente está seguro de que proviene de AS. Las marcas temporales le dan al cliente un nuevo origen de tiempos y un intervalo de operación, dentro del cual tiene validez el ticket.

• **Mensaje 3: de C a TGS: { IDs, $Ticket_{TGS}$, $Autentificador_C$ }**

Con este mensaje el cliente le solicita al TGS el ticket de servicio.

$ID_S$ : Se le comunica al TGS que el cliente requiere los servicios del server cuya identidad es $ID_S$.

$Ticket_{TGS}$ : Gracias a este ticket el TGS reconoce como auténtico al cliente que demanda los servicios.

$Autentificador_C = Ek_{c,TGS}[ID_c, AD_c, TS_3]$: este campo asegura al TGS que el ticket pertenece al cliente. Está compuesto por la identidad del cliente la dirección de la máquina del cliente y un sello temporal que indica el momento en que se generó este campo. No debemos confundir la dirección de una máquina con su identidad, a su vez un ticket es el que autentifica al cliente y el otro es el que le autoriza a hacer uso de los servicios demandados del servidor.

**Mensaje 4: de TGS a C: {$Ek_{c,TGS}$[ $K_{c,S}$, $ID_S$, $TS_4$,Tickets]}**

Con este mensaje El TGS devuelve al cliente el ticket de servicio requerido. Mensaje cifrado con la clave que comparten cliente y TGS.

$K_{c,S}$ : Clave de sesión que el cliente empleará en sus comunicaciones con el servidor.

$ID_S$ : Identidad del servidor demandado.

*Un protocolo que intente garantizar la seguridad en un entorno abierto puede llegar a ser muy complicado*

$TS_4$ : Informa al servidor del momento en que se generó el ticket.

Tickets = $Ek_S$ [$K_{c,S}$, $ID_c$, $AD_c$, $ID_S$, $TS_4$, $Lifetime_4$]. Este es el ticket con el que el cliente accederá al servidor, acreditándole como que tiene derecho al servicio demandado.

$Ek_S$ : El ticket es cifrado por una clave que sólo conocen el TGS y el server, con el objeto de evitar falsificaciones.

$K_{c,S}$ : clave de sesión compartida por cliente y servidor

$ID_c$ : Identidad del cliente.
$AD_c$ : dirección del cliente.

$ID_S$ : Identidad del servidor.

$TS_4$ : Momento en que se generó el ticket.

$Lifetime_4$ : Caducidad del ticket.

Con este mensaje el cliente tiene por fin un ticket que le autoriza ante el servidor como usuario de pleno derecho del servicio requerido. Como el otro, viene protegido por sellos temporales y cifrado por una clave que sólo conocen servidor y TGS.

• **Mensaje 5 de C a S: {Tickets, $Autentificador_C$}**

El cliente solicita el servicio al servidor.

$Ticket_S$: es el explicado antes.

$Autentificador_C$: $Ek_{cv}$[ $ID_c$, $AD_c$, $TS_5$ ]

Gracias a este campo el servidor se asegura de que el ticket pertenece al cliente. Los campos que lo componen son fácilmente reconocibles. Cifrado con la clave común de cliente y servidor, identidad y dirección del cliente y un sello temporal indicando el momento en que fue generado.

• **Mensaje 6: De S a C: {$Ek_{CS}$ [$TS_5$ + 1]}**

Mensaje opcional con el que el servidor se autentifica ante el cliente, para lo cual manda $TS_5$ +1, garantizando la "frescura" del mismo.

Vemos pues lo complicado que puede llegar a ser un protocolo que intente garantizar la seguridad en un entorno abierto. Mediante cifrados, claves, sellos temporales, tiempos de vida y un largo etcétera. Kerberos nos proporciona autentificación, acceso a servicios según los usuarios tengan derecho, todo ello de forma transparente al

usuario: sólo con abrir una sesión y solicitar el servicio, el software se encarga de gestionar todo lo necesario para atendernos.

## Versión 5

A pesar del aspecto del protocolo en la versión cuatro, éste tenía muchas dependencias y limitaciones. Dicha versión fue la primera en emplearse y las versiones de la una a la tres solo fueron parte del proceso de desarrollo. Las principales deficiencias de la versión cuatro son las siguientes :

1.- La versión cuatro emplea como cifrador un DES, en la cinco se puede especificar el algoritmo de cifrado.
2.- Dependencia de las direcciones IP. En la cinco esto no ocurre y podemos emplear otro tipo de direcciones, según el protocolo empleado.
3.- El orden de los bytes quedaba un poco al gusto del usuario; en la nueva se emplea el lenguaje ASN.1 (Abstract Syntax Notation Uno y la codificación BER ( Basic, Encoding, Rules ), lo que le hace seguir los estándares de codificación.
4.- Mayor flexibilidad a la hora de establecer la duración de los tickets.
5.- Autentificación avanzada. Gracias a esta nueva capacidad del protocolo podemos ceder derechos a terceros sin necesidad de que éstos tengan que autentificarse.
6.- La autentificación entre entornos distintos se hace de manera más eficiente

En cuanto a los mensajes, se han añadido algunos campos y otros se han suprimido. Para los mensajes uno y dos pertenecientes a diálogo de autentificación, tenemos a Realm el cual indica el dominio del servidor o servidores TGS, y AS se usa para cuando solicitamos servicios de servers que normalmente son externos al nuestro.

En el campo Options se ponen unos determinados flags para optar por unas u otras características del protocolo. El campo Times nos aporta mayor flexibilidad a la hora de fijar la duración de los tickets y, por último, Nonce es un campo donde se pone un número aleatorio para garantizar la frescura de la respuesta.

En el diálogo de obtención del ticket de garantía, mensajes tres y cuatro, se han añadido campos de opciones para el ticket y los números aleatorios. Según se pongan unos u otros, los tickets y números aleatorios tendrán unas características diferentes.

Por último, en el diálogo de cliente servidor, mensajes cinco y seis, se han añadido dos campos más: el campo Subkey que da opción a que se use o no el cifrado entre ambos, lo que, si no estamos en ninguna operación sensible, acelera las comunicaciones al evitarnos cifrar los mensajes; y el otro campo, Número de secuencia, se puede emplear opcionalmente para evitar duplicados de mensajes. Los mensajes quedarían compuestos por los siguientes campos, donde sus nombres tendrían los mismos significados que en la versión 4.

• Mensaje 1: de C a AS: {Options, IDc, Realmc, IDTGS, Times, Noncei}.

• Mensaje 2: de AS a C: {Realmc, IDc, TicketTGS, Ekc[Kc,TGS, Times, Noncei, Realmc, IDTGS ]}.

TicketTGS = EkTGS[Flags, Kc,TGS, Realmc, IDc, ADc, IDTGS, Times]

• Mensaje 3: de C a TGS: {Options, IDS, Times, Nonce2, TicketTGS, AutentificadorC }

Autentificadorc: Ekc,TGS[IDc, Realmc, TSi ]

$Ticket_{TGS} = Ek_{TGS}$[Flags, $K_{C,TGS}$, $Realm_C$, $ID_C$, $AD_C$, $ID_{TGS}$, Times]

• Mensaje 4: de TGS a C: {RealmC, IDC, TicketS EkC,TGS[KC,S, Times, Nonce2, RealmV, IDS ]}

$Ticket_V = Ek_S$ [Flags, $K_{C,S}$, $Realm_C$, $ID_C$, $AD_C$, Times]

• Mensaje 5 de C a S: {Options, $Ticket_S$, $Autentificador_C$}

El cliente solicita el servicio al servidor.

$Ticket_S = Ek_S$ [Flags, $K_{C,S}$, $Realm_C$, $ID_C$, $AD_C$, Times]

$Autentificador_C : Ek_{CS}$ [ $ID_C$, $Realm_C$, $TS_2$, Subkey, Seq# ]

• Mensaje 6: De S a C: {EkCS[ TS2 Subkey, Seq# ]}

## Conclusiones

Hemos visto los problemas de seguridad en cuanto a la resistencia que presentan los protocolos a los ataques de que son objeto, debido a la naturaleza abierta de la red. Nos hemos dado cuenta de lo complejos que pueden llegar a ser los protocolos al igual que sorprendentes y rebuscados que puedan ser los ataques. Lo que nos lleva a la conclusión de que la seguridad mas que una realidad, es un desafío de todos los días.

## Bibliografía


• RFC 1510.
• "Network and Internetwork Security ", Autor William Stallings, editorial Prentice-Hall.
• "Computer Networks", Autor Andrew S. Tanenbaum, editorial Prentice-Hall.


**Fe de erratas**

En el artículo correspondiente al número 35 de Sólo Programadores sobre criptografía, siempre que aparece en el texto el carácter Â se corresponde con Σ (sumatorio).

# La función de evaluación

*Eugenio Castillo Jiménez*

Comencemos por discutir el sistema de representación de un tablero de ajedrez tomando como muestra los programas públicos existentes, por un lado tenemos el sistema TPChess que usa un recuadro de 8x16 bytes (en realidad no define en el array los últimos 8 bytes), de los que sólo se utiliza un subrecuadro de 8x8, por otro lado tenemos el GNU y CRAFTY que usan uno convencional de 8x8.

## Estructuras de datos

En ambas estructuras se insertan los datos correspondientes al color y el tipo de pieza que hay sobre una casilla (en realidad se utiliza un tablero por tipo de dato), Existe también un tercer dato empleado en algunos programas que es el indexado de la pieza sobre el tablero en la lista de piezas por bando.

Veamos un ejemplo en Pascal:

```
type  { Casillas: a1=0, b1=1,..,
        a2=$10,..., h8=$77 }
TipoCasilla =   0..$77;
{ Las 64 casillas, aunque sean 120 las definidas (el
        8x16 - 8 comentado anteriormente) }
TipoColor= (Blancas, Negras);
TipoPieza=(Vacio,Rey, Dama, Torre, Alfil, Caballo,
        Peon);
TipoIndex   = 0..15;
{ Indexado sobre TabPiezas }

var
TableroPiezas : array[TipoCasilla] of
        TipoPieza;
TableroColor : array[TipoCasilla] of
        TipoColor;
TableroIndex : array[TipoCasilla] of
        TipoIndex;
{ TabPiezas contiene las casillas de las piezas y su
        tipo. La primera pieza suele ser el rey }

TabPiezas :
 array[TipoColor,TipoIndex] of
record
        ICasilla : TipoCasilla;
        IPieza:  TipoPieza;
end;
```

Estas estructuras de datos suelen estar íntimamente ligadas a los generadores de movimientos y a las rutinas que efectúan y retroceden las jugadas de las piezas sobre el tablero. Podemos decir que cualquier sistema que diseñemos debe de tender a dar la máxima velocidad e información posible.

El tablero 8x16 fue ideado por State y Atkin en 1977, su ventaja radica en la aceleración del generador de movimientos.

Una buena parte de los generadores se basan simplemente en vectores de movimiento, de este modo el alfil con un incremento +17 movería hacia arriba-derecha, el problema se encuentra en que

La denominada función de evaluación es uno de los componentes claves de un programa de ajedrez. Por lo general su resultado se da en un integer y es el producto del estudio de una serie de condiciones. Hasta hace poco se tendía a imitar las evaluaciones humanas usando conceptos humanos. Veamos cuáles son las tendencias actuales.

hemos de chequear los límites del tablero para que el alfil no pase directamente desde la última casilla de la derecha a la primera de la izquierda de la siguiente línea (esto ocurriría en un tablero 8x8 con un incremento de +9). Con el tablero 8x16 al "salirnos" por el borde entramos en un territorio de nadie que sería fácilmente detectado con un **"casilla and $88"**. Ejemplo:

```
casilla := 7;

while (true) do
begin
  casilla := casilla + 17;
  if (casilla and $88) <> 0 then
    goto findir17;

 if  (TableroPiezas [casilla] <> Vacio) then
  begin
    if (TableroColor [casilla] = Contrario) then
      LincaMov; {añadir captura y saltar a la si-
      guiente dirección}
    goto findir17;
  end
else
  begin
    {añadir movimiento}
    LincaMov;
  end;
end;

findir17:
```

## El tablero 8x8

Por otra parte, los seguidores del tablero 8x8 han ideado sistemas de generación de movimientos realmente sofisticados. El GNU, por ejemplo, almacena todos los movimientos posibles para cada pieza en cada una de las casillas del tablero.

Evidentemente, este sistema ocupa una cierta cantidad de memoria que ya comenzó a ser razonable en los ordenadores de más de 512K Ram. El código que efectúa este trabajo es realmente elegante (el ejemplo es válido para todas las piezas excepto peones), veámoslo:

```
struct sqdata {
    short nextpos;
    short nextdir;
};

struct sqdata posdata[8][64][64];

/*posdata[tipopieza][casillaorigen]
[casilladestino] ejemplo;*/
```

el primer movimiento para un alfil situado en la casilla e8 sería: **posdata [alfil][e8][e8].nextpos**, si en f7 hubiera una pieza bloqueando la posición **posdata [alfil][e8][e7].nextdir** nos daría la casilla inicial de la siguiente dirección. En caso de que nos topásemos con un borde tanto **nextpos** como **nextdir** nos darían la casilla válida, de hecho aquí no es necesaria esta comprobación. El final de la generación se da cuando la casilla destino es la misma que el origen.

```
/*
p : puntero de posdata;
u : variable casilla;
sq : casilla original de partida;
*/

  p = posdata[piece][sq];
  u = p[sq].nextpos;
  do {
   if (color[u] == neutral)
      {
     LinkMove(ply,sq,u); //LinkMove inserta el
     //movimiento en una buffer
     u = p[u].nextpos;
   }
   else {
     if (color[u] == xside)
          LinkMove(ply,sq,u);
     u = p[u].nextdir;
   }
  } while (u != sq);
```

Un tablero de 8x8 reúne unas condiciones idóneas para almacenar determinados tipos de información como pudieran ser máscaras de bits. Si quisieramos conocer el área que pudiera barrer/controlar una pieza como una torre sobre el tablero, ésta sería fácilmente representable con ocho bytes, constituyendo cada byte un línea del tablero y cada bit un escaque.

Esta circunstancia ha sido aprovechada con bastante buen tino por los programadores del CRAFTY, los cuales hicieron el programa para un procesador 64 bits que manejaba los tableritos 8x8 "de una tajada". Utilizando este sistema podemos conocer casi de inmediato el número de casillas que amenaza una pieza sumando los bits a uno que posee su máscara, esto se puede hacer fácilmente almacenado previamente el número de bits a uno de un entero en memoria (65536 casos posibles) y sumando los cuatro enteros que constituyen una máscara de tablero.

*Utilizando el sistema del CRAFTY podemos conocer de inmediato el número de casillas que amenaza una pieza*

El GNU calcula esto mismo generando los movimientos de la pieza uno a uno. Sin embargo, a todas luces este método nos parece más rudimentario y costoso en tiempo.

## Utilización de máscaras

Otra circunstacia favorable a los usuarios de máscaras reside en el hecho de que mediante operaciones **and** pueden obtener informaciones valiosas, por ejemplo, se puede saber si una pieza amenaza a otra haciendo directamente un **and** entre la máscara de la pieza atacante y la máscara de la casilla de la pieza amenazada, si la operación da como resultado cualquier valor distinto de cero entonces esto significa que existe contacto.

Incluso es posible determinar si una pieza que no amenaza a otra puede

hacerlo con un sólo movimiento siguiendo un sistema muy parecido al anterior. Por contrapartida el generador de movimientos mostrado en los fuentes del CRAFTY es sólo utilizable en ordenadores de 64 bits con operaciones especiales a nivel de bits.

## El tablero Atak

El GNU utiliza otros recursos para realizar el análisis, cabe destacar el de las zonas de control representados en las tablas **atak**, en ellas se introducen los tipos de piezas y el número de éstas que llegan a cada escaque. Veamos con un ejemplo cómo se introduciría la información en las tablas:

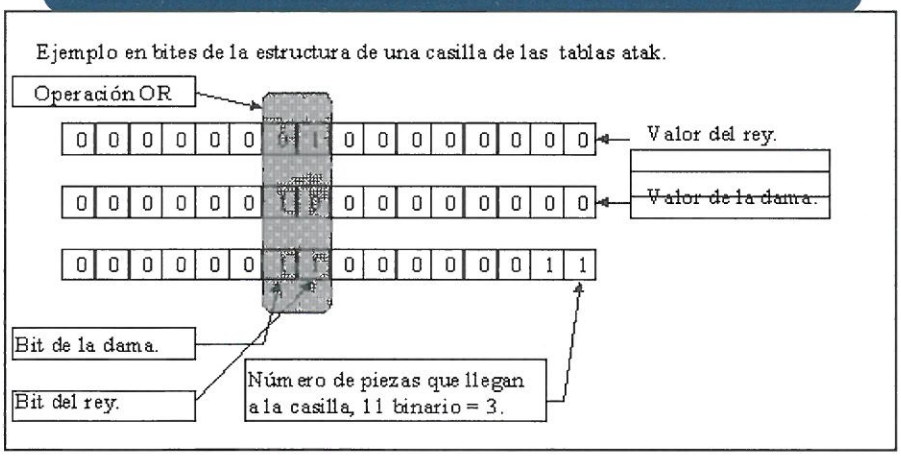
```
#define ctlP 0x4000    //control de peón
#define ctlN 0x2800    //control de caballo
```

*Las tablas atak contienen los tipos de piezas y el número de éstas que llegan a cada escaque*

Cada pieza posee un código de relación sobre el tablero (peón=0x4000,alfil=0x1800,rey=0x0100,etc..). Estos códigos sólo poseen valores superiores al 255 por lo que la parte baja de cada short permanece libre, aquí cada pieza que pasa/amenaza la casilla es contada, de tal modo que el short queda con su parte alta conteniendo el tipo de piezas que llegan a ella y en la parte baja el número de éstas sin discriminar tipos. Este volumen de datos posee un gran interés para realizar evaluaciones de alto nivel, un ejemplo de información deducible podría ser el conocer si uno de los bandos domina una casilla importante simplemente comparando **atak [white] [h7] > atak [black] [h7]** (el resultado de esta comparación no es del todo exacto pero sí nos da una referencia altamente aproximada), también nos podría decir si una pieza puede ser capturada si se dirige a una casilla determinada, si un alfil amenaza llegar a la casilla h3 **atack [black] [h3]& ctlB**, también existe una rutina que evalúa el "peligro" de ataque al rey chequeando los posibles jaques de las piezas contrarias. Como queda demostrado, el uso del tablero **atak** es fundamental.

La contrapartida de este sistema radica en el hecho de que tras cada movimiento efectuado los valores de **atak** cambian y es preciso regenerarlos con la consiguiente pérdida de tiempo. Actualmente se han desarrollado sistemas (no publicados pero sí mencionados) que aprovechan los valores anteriores para modificar tan sólo los necesarios (aquellos que ha cambiado puesto que cada movimiento sólo modifica parcialmente la tabla), con ello se consiguen grandes ahorros de tiempo.

Figura 1




```
#define ctlB 0x1800   // alfil
#define ctlR 0x0400   // torre
#define ctlQ 0x0200   // dama
#define ctlK 0x0100   // rey
#define ctlBQ 0x1200 // alfil dama
#define ctlBN 0x0800 // alfil caballo
#define ctlRQ 0x0600 // torre dama

void ataks ()
{
  short int *a; //puntero hacia la tabla atak
  short u; //casilla
  a = atak [blanco];
.....
  c = control [pieza]; //control contiene el código
      // ctl arriba indicado para cada pieza

  do {
      a [u] =  ++a[u] | c; //suma y or de c sobre
          //la casilla controlada
      u =  p[u].nextpos;
   } while (u != sq);
  .....
}
```

## Mejoras a los Tableros atak

El sistema de los tableros **atak** es manifiestamente mejorable, existen circunstancias fortuitas que pueden ser aprovechadas, entre ellas el hecho de que "sólo" existan 16 piezas por bando, osea el número de bits que ocupa un **integer**. De este modo se podría definir otro tablero por bando, por ejemplo:

```
short CtrlBlanco [64],CtrlNegro [64];
```

y si asignamos un bit (del 1 al 128) a cada pieza, al insertar los valores en los tableros **atak** añadiríamos una operación or :

```
CtrlBando [u] |= BitPieza;
```

de modo que **BitPieza** es el short respectivo de cada pieza.

Realizando estas operaciones ya disponemos de una nueva información valiosa, uno de los primeros campos a los que se podría aplicar se encontraría en la generación de los movimientos para las búsquedas de tranquilidad, aquí sólo se consideraban los movimientos de capturas. El TPChess realiza este proceso comprobando una a una si sus piezas son atacadas por otras enemigas (la cantidad de preguntas a realizar sería equivalente al número de piezas propias por las del contrario exceptuando los reyes como pieza capturable), el GNU en cambio genera todos los movimientos de sus piezas descartando las que no son capturas. Con el nuevo tablero **CtrlBando** todo ello es innecesario, sólo preguntaríamos pieza por pieza del bando contrario si el **CtrlBando** es distinto de cero, si lo fuera bastaría con un código sencillo para reconocer los atacantes que pueden capturar.

Para posteriores desarrollos de movimientos selectivos (aquí sólo se generan los movimientos considerados como necesarios o interesantes) también son muy importantes estos tableros.

Otro aspecto mejorable de las tablas **atak** reside en la representación de las amenazas sobre una casilla, existen sistemas mejorados que dan más exactitud a los posteriores cálculos. Veamos un ejemplo:

```
#define VP 64    //control de peón
#define VN 16    // control de caballo
#define VB 16    // alfil
#define VR 4     // torre
#define VQ 1     // dama
#define VK 1     // rey

char VBlanco [64],VNegro [64];

void ataks ()
{  short int *a; //puntero hacia la tabla atak
   short u; //casilla

   a = VBlanco;
   .....
   c = Hcb [pieza]; //Hcb tiene el código V arriba
       indicado para cada pieza
   do {
      a [u] += c; //suma del valor de ataque
      u = p[u].nextpos;
   } while (u != sq);
   .....
}
```

Mediante este método (usado por algunos programadores españoles entre los que me incluyo) podemos representar en ocho bits de una forma ideal la fuerza ejercida por un bando sobre una casilla. El valor para el alfil y el caballo es el mismo puesto que su valor material es muy aproximado (otro tanto sucede con la dama y el rey aunque este caso sea más discutible), el orden de fuerza es mínimo para el rey y máximo para el peón (muy parecido al del GNU). La bondad de este sistema radica en que dado que podemos representar un bando en un byte y ambos bandos en un integer, resulta que se reducen todas las posibles combinaciones a 65536, almacenándolas en memoria con su resultado respectivo, cada vez que deseemos conocer el signo de una captura o quién es el bando dominante de una casilla simplemente habremos de leer el valor de la tabla previamente calculada.

Para casos excepcionales como más de tres caballos o tres alfiles o tres torres y para más de dos damas pueden darse situaciones de valores falsos. Lo cierto es que estas posiciones sólo se dan en ocasiones muy contadas (más de una coronación y además ninguna captura de piezas similares anteriores).

## La rutina de Evaluación

La rutina de evaluación estudia cada pieza sobre el tablero y según el tipo de pieza se evalúan sus conceptos pertinentes. Por lo general, cada concepto suele tener un valor positivo o negativo que se suma al total por bando, la mayor parte de estos valores oscila entre -30 y +30. La diferencia entre las sumas por bando es el valor de la posición. Veamos los conceptos y el modo en que se evalúan:

1. **Material**, valor de las piezas. Podemos decir que es, en principio, uno de los criterios más importantes. Cabe destacar que en algunos programas el valor de las piezas cambia según la fase de la partida en que nos encontremos, por ejemplo en los finales los peones aumentan de valor. En el programa GNU existe una variable llamada **stage** que indica la fase del juego (apertura, medio juego y final), su valor se determina a partir de la cantidad de material que existe sobre el tablero.

Los valores de las piezas según GNU:

```
#define valueP 100    //peón
#define valueN 350    //caballo
#define valueB 355    //alfil
#define valueR 550    //torre
#define valueQ 1100   //dama
#define valueK 1200   //rey
```

Este valor es muy parecido al que le conceden los jugadores humanos a las piezas.

2. **Mobilidad**, algunas piezas (torres, alfiles y a veces damas, según los programas) desarrollan un área de acción, según el CRAFTY sólo se consideran las casillas vacías a las que puede acceder la pieza. Según el GNU, el campo se amplia considerando temas de clavadas y descubiertas. Los valores que se dan oscilan entre -2 para mobilidad 0 y 16. La dama no se suele incluir en este grupo porque puede llegar a producir valores falsos sobre todo en la fase de apertura (precisamente uno de los primeros errores reiterados de los viejos programas consistía en sacar la dama al centro del tablero en las primeras jugadas, lo cual es antiposicional).

3. **Estructuras de peones**, seguramente el factor más importante

después del material. Existen muchos programas amateur que han perdido partidas sólo por tener malas evaluaciones de peones. Veamos algunas de las consideraciones más comunes:

3.1. **Peones aislados.** Existe un array de ocho bytes (dependiendo de los programas un byte siendo cada columna un bit) que indica el estado de ocho columnas (ocupadas o vacías). Un peón aislado es aquel que no posee peones en las columnas adyacentes y es considerado como una debilidad puesto que no puede ser apoyado por un compañero.

```
short AISLADO[8] =
{-12, -16, -20, -24, -24, -20, -16, -12};
```

En esta declaración de valores para peones aislados por columnas podemos ver que dependiendo de la columna (cercanía al centro) aumenta o disminuye el castigo.

3.2. **Peones doblados.** Con un contador de peones por columna sabemos la cantidad de peones que hay en una columna; dos o más peones en una columna reducen el valor material de los peones. Existe una penalización extra para los peones doblados y que a su vez estén aislados.

3.3. **Peones débiles.** La noción de peón débil es altamente original y determina muchas veces la fuerza de una estructura de peones. Se considera como peón débil aquel que no está apoyado por otro del mismo bando o que no tiene un compañero que apoye su avance (avance equivale a un paso adelante). Si el peón es considerado como débil se considera un factor agravante el número de piezas enemigas que lo atacan, otro agravante se considera el estar sobre una columna abierta. No se toman como peones débiles a los de **e4** o **d4** peones blancos, **e5** y **d5** para negras.

Por curioso que parezca este criterio tiende a hacer más armónicos los movimientos de los peones. Es de destacar que los programas que no poseen unos criterios definidos al respecto, tienden a producir estructuras débiles o a mover los peones de una forma descoordinada cuando no saben qué hacer.

3.4. **Obstrucción de los peones de 2D y 2R** mediante piezas u otros peones en 3D o 3R. Estos peones pueden actuar a modo de cuellos de botella en los movimientos del bando afectado si son bloqueados. Este bloqueo es penalizado.

3.5. **Peones pasados.** Se considera peón pasado aquel que se encuentra en una columna abierta (sin peones contrarios). El GNU hace un análisis "exhaustivo" (siempre mejorable) observando los ataques contrarios sobre las casillas que quedan hasta la octava fila.Si alguna de estas casillas es amenazada por un peón contrario el valor del peón pasado baja considerablemente (de hecho ya no es "pasado"). Si la amenaza fuera de otra pieza bajaría un grado, si está dentro del alcance del rey contrario también se produciría otra degradación, y, por último, otro factor a considerar es del bando que mueve (favorable si es el propio). Un peón pasado en buenas condiciones puede llegar a alcanzar el valor de una torre.

## Tabla 1

```
short PassedPawn0[8] =
{0, 60, 80, 120, 200, 360, 600, 800};

short PassedPawn1[8] =
{0, 30, 40, 60, 100, 180, 300, 800};

short PassedPawn2[8] =
{0, 15, 25, 35, 50, 90, 140, 800};

short PassedPawn3[8] =
{0, 5, 10, 15, 20, 30, 140, 800};

.....

if (PC2 [fyle] == 0)
//PC2 referencia del número de peones
//contrarios por columna, fyle es la columna
// del peón
{  if (side == white)
     r = rank + 1;
   else
   //se favorece el turno del negro, rank posee
  //la línea del peón, r retoma el valor
     r = rank;
      // el rey contrario llega al peón
   in_square = (row (wking) <=
            r && distance (sq,wking) < r + 1);
   if (a2 == 0 || side == black)
     e = 0; //e es la variable que contiene el
            //grado del "peón pasado"
   else
     e = 1;

   for (j = sq - 8;j >= 0; j-=8) //comprobación
// del estado del espacio entre el peón y el
// punto de coronación
     if (atk2[j]>=ctlP) //¿llega algún peón
//contrario a obstaculizarnos?
//atk2 es atak [white]
     { e= 2; break;}
      else if (atk2[j]>0 || color [j] != neutral)
        e = 1;
   //valoración del peón pasado según e
   if (e == 2)
    s += (stage *  PassedPawn3[7 - rank])/10;
   else if (in_square || e == 1)
    s += (stage * PassedPawn2[7 - rank])/10;
   else if (emtl[white]>0)
           s  +=(stage  *  PassedPawn1[7 -
1rank])/10;
   else
           s +=PassedPawn0[7 - rank];
  }
}
```

Obsérvese por las declaraciones la valoración del peón pasado por líneas, la 0 sería la óptima (sin oposición a la coronación). La elección del grupo viene determinada por la variable **e**. Veamos un ejemplo de la evaluación de un peón pasado negro en la tabla 2.

Se dan muchos casos de evaluaciones incorrectas de peones pasados tanto por exceso como por defecto, estos errores conducen a pérdidas de partidas en los finales. Uno muy común es el de sobrevalorar un peón pasado y débil que finalmente es capturado.

4. Ataque al rey.

4.1. **Estructura cercana de peones.** Se valora la existencia de peones cercanos al rey según el valor que contenga **stage**.

4.2. El denominado **ataque a la bayoneta**, que consiste en lanzar los peones contra la estructura del rey contraria. Este efecto se consigue fácilmente con un array de valores que premie la proximidad de los peones propios al rey contrario, el objetivo consiste en cambiar estos peones para abrir las columnas, un exceso de "premio por avanzar" puede producir un efecto negativo porque el ordenador prefiere conservar el peón "avanzado en vez de cambiarlo, es más, si ve que el rival se lo puede cambiar ni siquiera lo avanza.

Este tipo de ataque se da ante todo cuando los reyes están en lados opuestos y existe material razonable para lanzar ataque (damas y torres).

4.3. **Posibilidad de jaques.** Una vez producido el desarrollo de las piezas, el GNU realiza un barrido en el área del rey para ver los posibles jaques del contrario. Una situación desfavorable puede bajar la evaluación hasta en dos peones (200 puntos), por lo que esto permite a la máquina llegar a sacrificar un material equivalente para intentar atacar al rey. El resto de los sacrificios suelen ser combinativos y no valorativos (osea por evaluación).

4.4. **Área de control.** El aréa cercana al rey (todos sus cuadros adyacentes) es escrutada en busca de amenazas contrarias. Si el rey se encuentra excesivamente amenazado se producen castigos en la evaluación.

4.5. **Columnas abiertas.** Si el rey está sobre columnas abiertas o semiabiertas, dependiendo del valor de **stage** (citada anteriormente) cuanto más alto sea el valor material sobre el tablero mayor será el peligro a considerar.

*El desarrollo de una buena evaluación no es fácil, no consiste únicamente en dar valoraciones positivas o negativas*

Todos estos análisis en su mayor parte han sido posibles gracias a la existencia de las tablas **atak** o similares, existen otros muchos factores a considerar, por ejemplo los referentes a estructuras de peones tipo fiancheto cuando existen amenazas rivales sobre la columna de torre, etc. Lo importante es captar el espíritu de los análisis sencillos y adaptarlo a aquellos más complejos y específicos.

El desarrollo de una buena evaluación no es tarea fácil, no todo consiste simplemente en dar valoraciones positivas o negativas, muchas de estas valoraciones tienen con frecuencia efectos paralelos indeseables y que se nos van a dar en las situaciones menos previstas. Por lo general, las mejores ideas suelen ser también las más sencillas y las que tienden a relacionar el conjunto de la situación.

Un efecto especialmente difícil de conseguir es el hacer que la máquina siga un plan con varias fases en el que en un momento determinado los criterios de valoración cambian de una a otra fase. Si los casos en que se activan o desactivan los criterios no están claramente definidos en el tiempo, nos podemos quedar a la mitad de un plan, haciendo el esfuerzo realizado inútil. Un ejemplo ocurre cuando al programa no le interesa abandonar una de las fases porque al pasar a la siguiente fase la valoración de la posición es inferior (lo cual es casi siempre imposible o absurdo).

## Referencias y Bibliografía


- TPChess, programa público de Borland (1985).
- Crafty, programa público de Bob Hyat (entre otros).
- GNU, programa público de GNU, autor principal John Stanback, Free Software Foundation.
- Slate, D. and Atkin, L.R. (1977). Chess 4.5 - The North Western University Chess Program.
- Computer Chess Compendium, D. Levy (Batsford, London).
- Chess Skill in Man and Machine, Condon, J.H. and Thompson, K. (Springer Verlag).
- Computer Chess Compendium, David Levy (Batsford)
- How Computers Play Chess, David Levy and Newborn Monroe (Freeman).

# Visual C++ para Windows CE

*Jorge Figueroa*

HERRAMIENTAS DE DESARROLLO

Con la aparición desde hace unos años de los primeros PDAs y ordenadores de bolsillo, este sector del mercado tecnológico ha ido creciendo cada vez más y ha llegado un momento en que hay disponibles muchos dispositivos de mano que por falta de haber sido establecidas unas bases estándar, son totalmente incompatibles tanto entre ellos como con los ordenadores de sobremesa, con los que se supone tienen que poder intercambiar datos de forma sencillísima.

Dado que la posibilidad de poder intercambiar datos entre estos dispositivos y los ordenadores de sobremesa es, hoy por hoy, algo que se puede considerar imprescindible para los productos de estas características, Microsoft ha visto una oportunidad para su software, y ha aprovechado la puerta que había quedado abierta introduciendo una versión de su Sistema Operativo Windows para portátiles y dispositivos en general.

Portátiles de pocas prestaciones y recursos para que se convierta en un estándar y facilite la conectividad con los ordenadores basados en sus homónimos de sobremesa, consiguiendo de paso también una hegemonía total sobre este apartado tecnológico.

Además de todo lo dicho, junto con la presentación de un sistema operativo portátil estándar, se han definido también unas normas estándar de las características físicas y técnicas que ha de poseer un dispositivo de mano para poder considerarse lo que ahora llamamos un HPC (**H**andheld**PC**) y que hacen referencia a la cantidad mínima de memoria que han de tener, las resoluciones mínimas de pantalla, las funcionalidades mínimas y que funcionen a ser posible en el Sistema operativo Windows CE. (Ver lista de normas HPC en el listado1).

El resultado de todas estas nuevas normas y nuevo soft, es que desde finales del año pasado, todos los productos a los que consideramos HPCs que se están presentando tienen que cumplir estas nuevas especificaciones e incorporar Windows CE o algún sustituto similar para poder ser considerados posibles competidores del mercado de HPCs.

Ejemplos de últimos modelos de HPCs que ya están diseñados bajo estas normas son: Casiopeia, de la firma Casio, PC Companion de Compaq; HP300LX y HP320LX de la casa Hewlett Packard; Velo 1 de Philips, etc. La plataforma Windows CE ya se ha convertido en un estándar casi de facto desde el momento en que apareció y ya están empezando a aparecer decenas de programas shareware, freeware y de venta para dicha plataforma.

En el artículo se va a hablar sobre uno de los entornos de desarrollo que puede utilizarse como ejemplo para conocer la forma de programación que Microsoft propone para esta nueva plataforma, indicando

Debido a la gran evolución tecnológica que se está produciendo en estos últimos años, los dispositivos portátiles de grandes prestaciones se están empezando a hacer cada vez más populares. Como respuesta a ello, Microsoft ha presentado WINDOWS CE, el que podemos considerar el Windows de Mano para todos estos dispositivos.

qué requerimientos se precisan para iniciarse y poder empezar así nuestras propias aplicaciones WINDOWS CE.

Tan sólo mencionar que si el lector quiere contactar con el autor o buscar más información , puede hacerlo desde la WEB **www.arrakis.es/~erde** o escribiendo directamente la dirección de correo electrónico **erde@arrakis.es**.

## Requerimientos Hardware

Para poder instalar este entorno integrado de desarrollo y usarlo como una plataforma que resulte mínimamente productiva, se requiere un equipo hardware determinado. Las características más destacadas que podemos mencionar son:

**Listado 1. Tabla de características que han de cumplir los HPCs.**

-Disponer instalado Windows CE (o un sustituto similar).
-Tener un tamaño de bolsillo.
-Tener teclado tipo QWERTY.
-Pantalla a ser posible táctil con una resolución mínima de 480*240*2 bits de profundidad, (4 tonos de gris)
-Mínimo de 4Mb de memoria ROM actualizable.
-Mímino de 2 Mb de memoria RAM actualizable
-Disponer de un puerto de infrarojos IrDA estándar.
-Poseer un puerto serie.
-Tener como mínimo una ranura PCMCIA del tipo II
-Tener capacidad para reproducir sonido .WAV.
-Led indicador de estado.
-Enganche opcional para sincronización.
-Disponer de API's WIN32S, que incluye, TCP/IP, PPP, Winsock 2.0, RAS, DCC y TAPI.

## Cómo se programa para Windows CE

En su estrategia de desarrollo para este entorno, Microsoft ha diseñado una herramienta llamada Visual C++ SDK/DDK que no es más que un completo entorno de desarrollo de C++ que se instala sobre una versión que tengamos instalada del VISUAL C++ 5.00 de Windows.

Este entorno para Windows CE puede instalarse tanto sobre Windows 95 como sobre Windows NT, aunque para obtener todas las prestaciones (ver más adelante emulación) que ofrece el paquete, sería recomendable disponer del Windows NT.

1- Un ordenador con un microprocesador mínimo de 486 a 66 Mhz. Sin embargo para poder hacer alarde de todas las prestaciones del entorno y funcionar mejor, sería recomendable la utilización de un Pentium a 133 Mhz.

2- Tener instalados 24 o 32 MB de RAM en memoria.

3- Disponer de cien Megabytes de espacio libre en el disco duro.

4- Disponer de un CD-ROM que cumpla las especificaciones multimedia actuales.

5- Tener un monitor instalado VGA o SVGA.

6- Poseer un ratón compatible Microsoft Mouse.

## Requerimientos Software

A parte del hardware que requerimos, también necesitamos tener instalado un software mínimo para poder instalar el WCE VC++.

Los requisitos software son:

1- Disponer del sistema operativo Windows NT 4.0 si queremos aprovechar todas las características del producto o Windows 95 si somos un usuario medio.

2- Tener instalado (como ya se ha dicho) la versión 5.00 del Visual C++ para Windows, ya sea la versión Profesional o la *"Enterprise edition"*.

3- Tener instaladas las librerías WIN32S que normalmente tendremos como parte del sistema.

4- Disponer de, como mínimo, treinta Megabytes libres de espacio libre en el disco duro para poder instalar el nuevo programa.

## *El nuevo Visual C++ SDK/DDK se instala como un add-on sobre Visual C++ 5.0*

Si cumplimos todos los requisitos, tanto hard como soft, podemos comprar directamente el kit como producto de venta.

El SDK/DDK C++ cuesta aproximadamente unos 199 dólares estadounidenses (279 en Canadá), aunque también se puede bajar el programa completo a través de la WEB de Microsoft, tanto por HTTP como a través de FTP. Me gustaría advertir que el producto lo componen 20 MB a bajar y que puede

costar unas cuantas horas de conexión Infovía, si se da el caso de que no se corte la transferencia o que pase algo "extraño", como suele ocurrir demasiado a menudo.

Cuando tengamos todos los requerimientos hard y soft y hayamos adquirido el paquete Visual C++ para programar en Windows CE, ya casi podemos ponernos a programar.

## Características generales del Compilador

Como descripción inicial general del Visual C++ para Windows CE, decir que se trata de un "add-on" para el Microsoft Visual C++ 5.0. Los desarrolladores que lo usen podrán utilizar las librerías WIN32s para crear aplicaciones WINDOWS CE.

### *Visual C++ para Windows CE soporta las Microsoft Foundations Classes (MFC)*

Al estar el entorno totalmente integrado en el Sistema IDE de desarrollo Visual Studio, éste se puede beneficiar de las utilidades extras que el mismo incorpora, como son el depurador, el Spy, el Zoomin, el editor de registros, el monitorizador de memoria, etc.

Además de todo ello, Visual C++ para Windows CE soporta las Microsoft Foundations Classes (MFC) incluyendo en ello tanto las *Classwizard* como las *Appwizard*.

Como complemento a todo lo ya mencionado, este entorno añade utilidades específicas especiales para poder aprovechar las plataformas de destino (de los HPCs) que suelen estar basados en microprocesadores SH-3 de Hitachi o el MIPS R4000, ya que, como se sabe, los dispositivos de mano suelen estar basados en los más diversos microprocesadores y su hardware no tiene nada que ver con el de las plataformas de sobremesa.

## Componentes del Compilador

A continuación se comentan los componentes de este nuevo sistema operativo:

1- Se integra totalmente en el Visual Studio 97, con lo que no tendremos que adaptarnos al programa puesto que ya conocemos (se supone) el entorno de desarrollo del Visual C++ de plataforma PC.

   Además, como hemos dicho, al estar integrado como una extensión del sistema, las demás utilidades también podemos aprovecharlas en beneficio de la programación para Windows CE. (Spy, Visores de objetos, visores de memoria, ...).

2- Posee un emulador integrado para poder ejecutar las aplicaciones que desarrollemos en este compilador sin tener que disponer del Handheld PC ni tener tampoco en consecuencia que descargar el ejecutable en él para poder probarlo en el propio HPC, con lo que se consigue una mayor velocidad de desarrollo de aplicaciones en las fases de compilacion-testeo-depuración..

3- Dispone de soporte integrado para poder usar las MFC en los propios fuentes, ya que Windows CE soporta las Microsoft Foundation Classes. Ello, conlleva, como ya hemos dicho, que podemos usar tanto las AppWizard y las ClassWizard, que funcionan con las librerías MFC.

4- Dispone de un sistema automatizado de descarga de los programas que compilamos en el HPC de destino usando solamente un cable, con lo que todas las operaciones se facilitan enormemente. Dicho esto, queda claro que no se requiere para trabajar con el PC y el Handheld de ningún hardware intermediario. Lo único que tenemos que hacer es conectar ambos dispositivos mediante el puerto serie al puerto COM (cable que viene incluido ya normalmente con el propio HPC cuando lo compramos) , ya que todas las operaciones de control las dispone el propio SDK en un API para poder realizar todas las operaciones automáticamente.

5- Se dispone de un depurador remoto que es una heramienta muy potente y eficaz que se convertirá en imprescindible para el programador e incluye las funcionalidades típicas, como los DATATIPS, menús de información, apertura de ventanas de monitorización locales, colocación de puntos de ruptura, control de la ejecución del programa y visor del estado de los registros.

6- El Kit viene acompañado de un pequeño lote de fuentes para ayudar en el aprendizaje de su programación, incluyendo exhaustivos ejemplos tanto en C, como en C++ y de MFCs, con lo que el programador podrá ver rápidamente las similitudes entre los programas escritos para Windows y los creados para WINDOWS CE y también las diferencias básicas, así como las nuevas tecnologías que hay disponibles en este sistema operativo y que deben conocerse para poder sacar el mayor partido posible del mismo.

7- Al ser un SDK/DDK, posee todo lo necesario como son las cabeceras (headers) de las librerías que incor-

pora, las librerías que contienen todas las funciones, ejemplos y utilidades en el DDK.

8- Como punto final, todo el Kit viene acompañado de una completísima documentación On-line que incluye capítulos sobre programación para Windows CE, la forma de exportar el código ya existente, cómo usar el API para controles remotos, el sistema que usa Windows CE para el transporte de MAILs y otras características que son especiales (específicas de este sistema operativo).

9- El compilador está optimizado y basado en las más modernas tecnologías de compilación, lo cual es un hecho fundamental para poder programar aplicaciones que deberán funcionar en sistemas con especiales limitaciones de memoria (recordar que en los HPCs, la memoria instalada suele ser de dos o cuatro MB, sin contar la que usa WINDOWS CE para su uso propio).

*Las plataformas destino por ahora se basan en dos microprocesadores diferentes*

Además de todo ello, como las plataformas destino pueden estar basadas en dos (de momento) microprocesadores diferentes (SH-3 y MIPS R4000), el compilador tiene librerías de todo el API optimizadas especialmente para cada versión de dichas CPUs. De esta forma se aprovechan al máximo las posibilidades de cada chip.

Un ejemplo práctico donde ello queda demostrado es que todos los programas disponibles para WINDOWS CE, ocupan menos memoria en cuanto a kilobytes se refiere, siempre en los binarios que corresponden a los programas para el microprocesador SH-3, fruto ello de que el compilador aprovecha las posibilidades de cada uno y que los binarios resultantes son totalmente diferentes.

10- Las librerías estándar WIN32S, que son unas quinientas llamadas de funciones en total, están casi todas soportadas por el Windows CE, excepto algunas puntuales que incorpora Windows CE exclusivas y que están diseñadas para aprovechar características especiales que ofrecen algunos dispositivos móviles como son algunas para el soporte de bases de datos, APIs para control remoto, etc.

Además, Visual C++ para Windows CE soporta completamente el ANSI C++ de Microsoft. Todo ello, junto con la compatibilidad ya mencionada del MFC , hará que los desarrolladores no tengan que realizar prácticamente una fase de aprendizaje para adaptarse al nuevo sistema operativo.

## Características propias del S.O.

Si queremos embarcarnos en la programación en C++ para este sistema operativo, debemos también, aparte de conocer el compilador que vamos a usar, saber las características intrínsecas del propio sistema en el que estamos desarrollando, para ser conscientes de lo que puede dar de si y, en consecuencia, conocer las características que podrán llegar a tener nuestras aplicaciones y saber qué proyectos podemos plantearnos.

## Datos generales

Para empezar, Windows CE es un sistema operativo de 32 bits (de ahí que soporte las WIN32S) y posee capacidad para ejecutar aplicaciones en multitarea y en multihebrado.

El sistema operativo completo, incluyendo el **H**ardware **A**bstraction **L**ayer (**HAL**, terminología usada también en la tecnología DirectX), el kernel de Windows CE, el GDI (**G**raphical **D**evice Interface), los ficheros del sistema y las bases de datos, ocupan menos de 1,5 MB de memoria, de los cuales tan sólo 700 Kb corresponden al propio kernel (núcleo del Windows).

El sistema operativo está diseñado como una plataforma totalmente abierta, y ello es así porque de esta forma puede adaptarse a los mil y un dispositivos portátiles que pueden llegar a instalarlo, cada uno de los cuales puede poseer características hardware de lo más diversas (teléfonos celulares , PDAs, HPCs, ordenadores de coche, televisores con internet, ...), por lo cual tiene una adaptabilidad al medio en que va a funcionar excepcional, que se traduce en cualquier cantidad de memoria instalada (desde los 2 MB), cualquier tamaño de pantalla, tanto táctiles como no, diferentes cantidades de bits por pixel de pantalla, cualquier microprocesador instalado, etc.

Está diseñado para poder funcionar desde la memoria ROM, puesto que todos estos dispositivos lo incorporan en este sistema de almacenamiento.

*Windows CE está diseñado para poder funcionar desde la memoria ROM*

Al estar destinado a funcionar en sistemas basados en baterías y pilas, posee funciones especiales para el ahorro de energía mientras funciona, ya que la duración de autonomía es siempre uno de los puntos críticos, alargando de esta manera al máximo la vida de las baterías.

Además de lo comentado, una de las características que lo va a hacer más

MASTERS DE INFORMÁTICA

# ENTORNO UNIX

## LA OBRA DE REFERENCIA MÁS COMPLETA Y ACTUAL

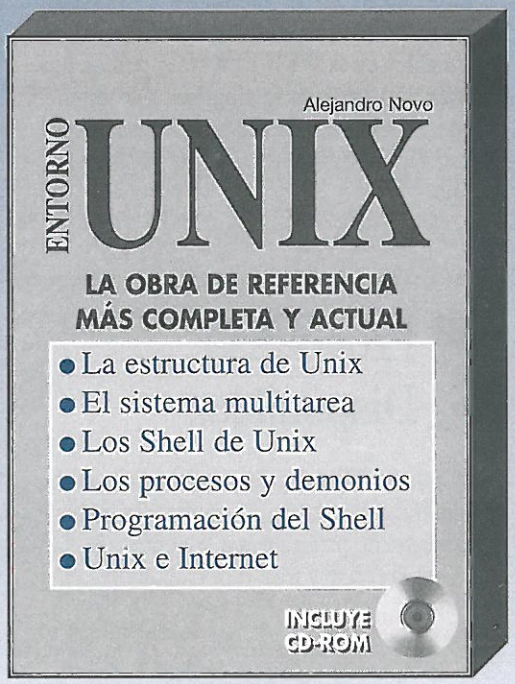
**EL ENTORNO UNIX es:**

- Uno de los sistemas operativos más extendidos.
- Es el candidato ideal para los servidores.

En esta obra podrá conocer en profundidad todas las particularidades de este sistema, contrastando la mayoría de sus variantes.

Tendrá asegurado un aprendizaje rápido y efectivo gracias al CD-ROM que acompaña el libro, el cual incluye:

- Todos los ejemplos que aparezcan en el texto,
- Programas de autor,
- Utilidades para Unix,
- Una versión completa de Linux.




**PRECIO: 4.995 Ptas.**

**LIBRO + CD-ROM**

ABETO EDITORIAL

c/ Aragoneses, 7
28108 Pol. Ind. Alcobendas (Madrid)
Tel.: (91) 661 42 11* - Fax: (91) 661 43 86

popular es la integración en el propio sistema operativo de la capacidad de conectarse a Internet para poder mandar y recibir E-MAILs, navegar por la World Wide Web, y otras funciones como mandar y recibir faxes.

## El futuro de Windows CE

Windows CE es muy joven, pero según la planificación de Microsoft, se usará también como plataforma en otras clases de dispositivos como pueden ser ordenadores de automóvil o incluso televisores de nueva generación con capacidades Internet que aparecerán en el mercado en breve.

Referente a las ampliaciones que sufrirá el sistema operativo, dado que está en su primera versión, seguramente en el futuro también dispondrá de algunas características de los WINDOWS actuales de PC (sobremesa), como son las tecnologías OLE, COM, ActiveX, DirectX, DDE, ODBC, sistemas de impresión, etc.

Las tecnologías en las que se encuentran investigando en la actualidad para introducir en este sistema operativo son JAVA y ActiveX, lo que parece indicar su importancia.

## Los chips soportados por WINDOWS CE

Como ya se ha mencionado, tanto el sistema como el compilador pueden hasta el momento funcionar tanto sobre el microprocesador SH-3 de la casa Hitachi, como en el MIPS R4000. Sin embargo, Microsoft ya ha informado de que en breve su sistema operativo soportará también los procesadores de AMD, ARM y Motorola, entre algunos otros.

*El sistema operativo WCE ocupa en total únicamente 1.5 MB*

De los dos chips mencionados soportados actualmente , el SH3 de HITACHI es el más potente que existe disponible hasta el momento. Es un chip de 32 bits, puede direccionar 4 Gb y además puede funcionar a 44 y 60 Mhz y dispone de cuatro u ocho Kbytes de memoria caché level 1, según el modelo de que se trate.

## Más información y ayudas

A parte de la infinidad de información que se puede obtener en la home page de Windows CE de Microsoft, hay disponibles en la WEB y en otros puntos de internet, tutoriales, programas de ejemplo y un sin fin de artículos dedicados al tema que seguramente le ayudarán mucho.

## Programas Disponibles

Windows CE es un estándar y como tal, significa que también empieza a haber decenas de programas a la venta, así como shareware y freeware disponibles para él.

Así, por ejemplo, podemos encontrar en la misma WEB de Microsoft dedicado al sector, una completa librería de direcciones internet clasificadas por temáticas de los lugares donde podemos conectar con WEB de diversas empresas dedicadas al desarrollo de productos Windows CE y comprar ON-Line programas, bajar versiones de demostración o incluso encontrar programas Freeware.

Dicho esto, podemos decir que el futuro de esta plataforma no es una incertidumbre, sino una realidad.

*Para aprovechar el emulador que incorpora, es necesario disponer de Windows NT*

## Conclusión

Como resumen, se puede decir que Windows CE ha unificado a todos los dispositivos móviles y que éstos se convertirán en verdaderas *extensiones* de los ordenadores de sobremesa.

Por otra parte, el hecho de que Microsoft haya creado el SDK Visual C++ Para Windows CE, ha hecho que miles de programadores expertos en el desarrollo de aplicaciones para Windows, puedan ampliar sus trabajos a una nueva plataforma ya que la fase de aprendizaje para adaptarse al nuevo sistema es prácticamente nula (tan sólo aprender unas cuantas funciones especiales del API WIN32s específicas de WCE).

En definitiva y a la vista de los resultados, podemos deducir que Windows CE ayudará a la creación de nuevos departamentos de programadores (nuevos puestos de trabajo) en muchas empresas que se dedicarán al desarrollo de aplicaciones para dispositivos móviles así como al nacimiento incluso de alguna que otra empresa totalmente nueva orientada específicamente al sector, ya sea tanto en la parte software como también en la parte hardware.

# Transferencia de datos con las MFCs. DDX y DDV

VISUAL C++

*Xesco Hernández*
*(xesco@redestb.es)*

Las MFCs incorporan una serie de funciones (DDX, DDV) para validar y transferir datos entre los controles de un diálogo y sus variables asociadas. Cuando usamos ClassWizard para asociar una variable a un control, elegimos control (Edit, Check, Radio, ListBox, ...) y seleccionamos uno de los tipos de datos predeterminados que se nos ofrecen (int, long, CString, ...). Hecho esto ClassWizard se encarga de añadir automáticamente a nuestra clase Diálogo una variable miembro del tipo seleccionado.

Por otra parte, la implementación que hacen las MFC de la clase CDialog conlleva que, de forma automática, se realice una transferencia de datos desde las variables miembro del diálogo hacia sus controles asociados en el momento en que se inicializa el diálogo, y otra transferencia, ahora desde los controles hacia las variables, cuando el diálogo se cierra pulsando el botón IDOK.

Si los DDX estándares proporcionados por las MFC y la implementación automática que éstas realizan de la transferencia de datos en un diálogo se adaptan a las necesidades de nuestra aplicación no necesitaremos escribir una sóla linea de código para transferir datos, ClassWizard lo hará por nosotros.

Pero hay situaciones en las que no sucede así, por ejemplo, cuando necesitamos asociar un control a una variable que no es miembro del diálogo, cuando queremos asociar un control con un tipo de dato definido por nosotros, cuando necesitamos hacer validaciones no previstas en las MFC o cuando la transferencia se debe producir no sólo al iniciar o cerrar el diálogo sino cuando lo deseemos. Al analizar estos casos nos vemos en la necesidad de adquirir un conocimiento más profundo sobre el funcionamiento interno de las transferencias de datos en las MFC para así poder adaptarlo a nuestras necesidades. Este artículo pretende proporcionar soluciones a algunas situaciones que se producen habitualmente en las que es necesario escribir código propio para la transferencia de datos.

## Transferencia Automática

Empecemos comprendiendo los detalles del funcionamiento de la transferencia de datos que implementan las MFC y ClassWizard. Cuando a partir de un recurso de diálogo usamos ClassWizard para crear nuestra clase derivada de CDialog, el Wizard escribe el código necesario para declarar la clase y escribe una versión redefinida de la función miembro de CWnd DoDataExchange(). En esta función el Wizard inserta automáticamente, para cada variable que asociemos a algún control del diálogo, una llamada a una función DDX. (Ver figura 1 y listado 1).

En el siguiente artículo trataremos un tema muy concreto de la programación con las MFCs, proporcionando soluciones a algunas situaciones que se producen cuando se necesita escribir código propio para la transferencia de datos.

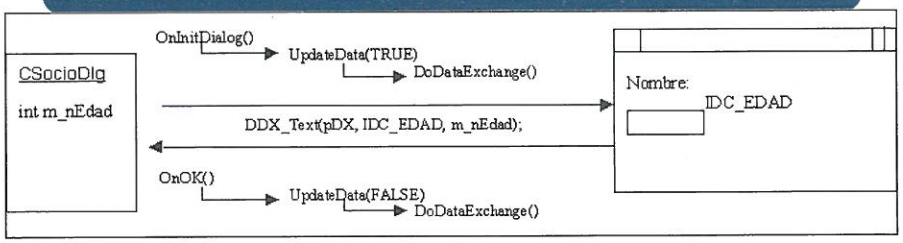
**Listado 1**

```
class CSocioDlg : public CDialog
{
...
public:

//{{AFX_DATA(CSocioDlg)
int m_nEdad;
//}}AFX_DATA

//{{AFX_VIRTUAL(CSocioDlg)

protected: virtual void
DoDataExchange(CDataExchange* pDX);
// DDX/DDV support
//}}AFX_VIRTUAL
...
};

void CSocioDlg::
    DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
//{{AFX_DATA_MAP(CSocioDlg)
DDX_Text(pDX, IDC_EDAD, m_nEdad);
//}}AFX_DATA_MAP
}
```

Nuestra clase derivada de CDialog ha heredado de ésta la función *OnInitDialog*. Esta función responde al mensaje WM_INITDIALOG, enviado por Windows justo antes de visualizar cualquier diálogo. La implementación que hacen las MFC de CDialog::OnInitDialog() inicializa los controles del diálogo y llama a UpdateData(FALSE), a su vez ésta llama a la función virtual DoDataExchange, que al estar redefinida en nuestra clase diálogo realizará las transferencias de datos que el Wizard escribió en nuestra clase. El uso de funciones virtuales redefinidas permite, pues, que el marco de la aplicación realice, de forma automática al inicializarse el diálogo, una transferencia de datos desde las variables hacia los controles de nuestra clase.

Siguiendo un planteamiento parecido, el mapa de mensajes de la clase CDialog implementada por las MFC tiene una entrada ON_COMMAND( IDOK, OnOK ) que llama a la función virtual CDialog ::*OnOK* en respuesta a la pulsación de un botón que tenga como identificador IDOK. Si nuestro diálogo tiene un botón con este identificador, cuando lo pulsamos, las MFC buscan una función de respuesta en el mapa de mensajes de nuestra clase de diálogo; al no encontrarla, buscan en el mapa de mensajes de la clase base, que llama a CDialog::OnOK(). La implementación que hacen las MFC de esta función llama a UpdateData(TRUE) y ésta llama a la función virtual redefinida DoDataExchange() de nuestra clase de diálogo. Es decir, la implementación que hacen las MFC de la clase CDialog y de los mapas de mensajes permite que el marco de la aplicación realice una transferencia de datos desde los controles hacia las variables, de forma automática, cuando el diálogo se cierra con IDOK.

**Figura 1**




La función *DoDataExchange()* es la responsable de la transferencia de datos en el diálogo. No debemos llamarla directamente, sino que es llamada por la función UpdateData(), miembro también de CWnd. Cuando deseamos transferir el contenido de las ventanas o controles hacia sus variables asociadas, llamamos a *UpdateData(TRUE)*, si deseamos transferir datos de las variables a los controles, llamamos a *UpdateData(FALSE)*.

UpdateData construye un objeto CDataExchange y llama aDoDataExchange pasándole un puntero a éste.

En toda transferencia de datos interviene un objeto de la clase CDataExchange. UpdateData() construye un objeto de la clase CDataExchange y llama a DoDataExchange pasándole un puntero a éste. La clase *CDataExchange* tiene dos atributos que nos informan sobre la dirección de la transferencia y sobre el diálogo en el que tiene lugar; mientras que *m_bSaveAndValidate* indica la dirección de la transferencia (TRUE si es desde controles hacia variables), *m_pDlg* es un puntero al diálogo en el que tiene lugar la transferencia.

## Variables externas al diálogo

Los DDXs creados con ClassWizard relacionan controles con variables miembro del diálogo. a continuación veremos cómo realizar transferencias con variables que no son miembros de la clase del diálogo.

Las funciones DDX transfieren datos entre una variable y un control en cualquiera de las dos direcciones posibles. Para realizar una transferencia necesitan como mínimo tres parámetros; un objeto de la clase CDataExchange, el identificador del control que interviene en la transferencia y una referencia a la variable que interviene en la transferencia.

Para transferir una variable cualquiera no tenemos más que pasar al DDX una referencia a la variable adecuada en lugar de una referencia a un miembro del diálogo.(Ver Listado 2.).

Supongamos que en el documento tenemos una variable miembro m_Socio de la clase CSocio. Queremos hacer un diálogo que muestre los datos de este socio y permita editarlos. Nuestro diálogo debe transferir datos entre m_Socio y los controles IDC_NOMBRE ,

## Listado 2

```
class CSocio
{
public:
        CString m_sNombre;
        int m_nEdad;
};
class CMiAppDoc : public CDocument
{
...
public: CSocio m_Socio;
};
class CSocioDlg : public CDialog
{
...
private:
        CSocio* m_pSocio;
}
void CSocioDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        ...
        DDX_Text(pDX, IDC_NOMBRE, m_pSocio->m_sNombre);
        DDX_Text(pDX, IDC_EDAD, m_pSocio->m_nEdad);
}
BOOL CSocioDlg::OnInitDialog()
{
        CMainFrame* pFrame = (CMainFrame*)(AfxGetApp()->m_pMainWnd);
        CMiAppDoc* pDoc = (CMiAppDoc*)(pFrame->GetActiveDocument());
        m_pSocio = &pDoc->m_Socio;
        CDialog::OnInitDialog();
        return TRUE;
}
```

## Listado 3

```
class CSocio

{
public:
        CString     m_sNombre;
        int         m_nEdad;
        CString     m_sNif;
        CString     m_sCodigo;
        BOOL        m_bConyuge;
        CString m_sNombreConyuge;
        int         m_nEdadConyuge;
};

class CSocioDlg : public CDialog
{
...
        public:
        CSocio m_Socio;
};

void
CSocioDlg::DoDataExchange
                (CDataExchange* pDX)
{

CDialog::DoDataExchange(pDX);
        ...
DDX_Check(pDX, IDC_CONYUGE,
        m_Socio.m_bConyuge);

if(pDX->m_bSaveAndValidate
        && !m_Socio.m_bConyuge)
        {

    m_Socio.m_sNomConyuge.Empty();
    m_Socio.m_nEdadConyuge = 0;
}
else
{

DDX_Text(pDX,
        IDC_NOMBRE_CONYUGE,
        m_Socio.m_sNomConyuge);

DDX_Text(pDX, IDC_EDAD_CONYUGE,
        m_Socio.m_nEdadConyuge);
}
}
```

IDC_EDAD, etc. En el diálogo declaramos un dato miembro del tipo puntero a CSocio, m_pSocio, que inicializaremos en OnInitDialog para que apunte al objecto m_Socio del mismo documento. CDialog::OnInitDialog() se encarga de llamar a DoDataExchange() y en los DDX asociamos los controles con los campos correspondientes del objeto m_Socio del documento, al que tenemos acceso a través del puntero.

Como ejemplo de las sentencias condicionales pondremos una situación en la que necesitamos escribir código nuestro en DoDataExchange(). Supongamos que a la clase CSocio le añadimos variables que indican si tiene o no cónyuge, el nombre y la edad de éste. (Ver figura 1). En la transferencia de datos queremos comprobar si se ha marcado el Check de cónyuge y si es así, transferir nombre y edad del cónyuge al objeto socio, pero si no se ha marcado el Check deseamos poner en blanco los datos referentes al cónyuge.

En este ejemplo las transferencias de datos se realizan con una variable m_Socio, miembro del diálogo, que es distinta a la que podamos tener en el documento. Al abrir o cerrar el diálogo debemos ocuparnos de inicializar o actualizar esta variable con los valores del socio del documento. (Ver listado 3.).

Usamos DDX_Check estándar para transferir datos en ambos sentidos entre m_bConyuge y IDC_CONYUGE. A continuación condicionamos la transferencia, según su dirección y si hay o no cónyuge. Si estamos transfiriendo desde controles hacia variables y se ha marcado el check como que no tiene cónyuge limpiamos los

datos del cónyuge, en cualquier otro caso transferimos los datos del cónyuge.

# DDX Propios

### • DDX_CUOTA

Cuando definimos tipos de datos propios, como clases, nos encontramos con la necesidad de implementar nuestras propias funciones DDX para transferir datos de este nuevo tipo. (Ver listado 4).

Supongamos que en el diálogo "Socio" necesitamos datos referentes a su cuota, como periodicidad (trimestral, semestral o anual) e importe. Debemos contemplar, además, el caso de socios que están libres del pago de cuota. Para ello declaramos una clase CCuota con un atributo entero m_nTipo (0. No tiene, 1. Trimestral, 2. Semestral, 3. Anual) y un long m_lImporte.

Un objeto de esta clase tendrá asociados cinco controles en el diálogo, cuatro RadioButtons y un Edit. Queremos poder transferir datos entre un objeto CCuota y sus controles llamando a una única función que se encargue de todo lo que sea necesario para ello. (Ver listado 4).

Todas las funciones DDX son funciones globales, usan el convenio de llamadass AFXAPI y no devuelven ningún valor. Para realizar la transferencia de un objeto CCuota nuestro DDX necesita como parámetros: un objeto CDataExchange, el identificador del primer control del grupo de RadioButtons, el identificador del Edit y una referencia a un objeto CCuota.

DDX_Radio se encarga de la transferencia entre el tipo de cuota y los radio buttons, tanto en el sentido de actualizar controles como en el de actualizar variables. Si estamos moviendo datos de controles hacia el objeto CCuota y se ha seleccionado tipo 0 "No tiene" nuestro DDX se encarga de asignar cero al importe, en cualquier otro caso usamos el DDX_Text estándar para transferir el importe.

### • DDX_NIF

En este ejemplo veremos cómo implementar una función de transferencia que no llama a ninguno de los DDX estándar de las MFC. Para ello nuestro código debe encargarse de leer y poner texto en el control cuando lo necesitemos mediante las funciones GetWindowText() y SetWindowText(). La transferencia de un NIF debe comprobar, además, si el formato del dato introducido por el usuario es correcto y fallar si no lo es, lo que nos permitirá ver cómo añadir control de errores a nuestros DDX.

Para comprobar errores disponemos de tres funciones miembro de CdataExchange: PrepareControl(), PrepareEditControl() y Fail().

*PrepareControl()* toma como argumento el identificador de un control, el último control que haya sido "preparado" recibirá el foco en caso de que llamemos a la función Fail(). *Fail()* lleva el foco al control previamente preparado con PrepareControl() y lanza una excepción. Cuando falla un DDX_ se sale de la función DoDataExchange(), sin realizar el resto de transferencias y UpdateData(TRUE), que llamó a DoDataExchange(), devuelve entonces FALSE.

**Listado 4**

```
class CCuota
{
public:
int m_nTipo;
// 0.No tiene, 1.Trimestral, 2.Semestral,
// 3.Anual.
LONG m_lImporte;
};
void AFXAPI DDX_Cuota(CDataExchange*
pDX, int nIDC_TIPO, int nIDC_IMPORTE,
CCuota& cuota)
{
DDX_Radio(pDX, nIDC_TIPO,
            cuota.m_nTipo);
if(pDX->
m_bSaveAndValidate&&cuota.m_nTipo==0)
        cuota.m_lImporte = 0L;
else
DDX_Text(pDX, nIDC_IMPORTE,
            cuota.m_lImporte);
}

class CSocio
{
...
public:
            CCuota m_Cuota;
};

void
CSocioDlg::
DoDataExchange(CDataExchange* pDX)
{

CDialog::DoDataExchange(pDX);
...
DDX_Cuota(pDX, IDC_CUOTA_NOTIENE,
IDC_IMPORTE, m_Socio.m_Cuota);
}
```

**Listado 5**

```
#define LON_NIF 9
BOOL ValidaNIF(CString nif)
{
   if(nif.GetLength() != LON_NIF)
            return FALSE;
   for(int i=0; i<LON_NIF-1; i++)
            if(!isdigit(nif[i]))
               return FALSE;
            if(!isalpha(nif[LON_NIF-1]))
               return FALSE;
            return TRUE;
}

void AFXAPI DDX_NIF(CDataExchange*
pDX, int nIDC, CString& nif)
{
  pDX->PrepareEditCtrl(nIDC);
  CWnd* pControl = pDX->m_pDlgWnd->
               GetDlgItem(nIDC);
  if(pDX->m_bSaveAndValidate) {
     pControl->GetWindowText(nif);
  if(!ValidaNIF(nif)) {
     pDX->m_pDlgWnd->MessageBox("NIF
incorrecto", "Error",
          MB_ICONEXCLAMATION);
     pDX->Fail();
   }
}
else
  pControl->SetWindowText(nif);
}

void
CDlg1::DoDataExchange(CDataExchange*
pDX)
{

CDialog::DoDataExchange(pDX);
            ...
DDX_NIF(pDX, IDC_NIF,
            m_Socio.m_sNif);
}
```

Utilizaremos una variable CString para guardar el NIF. Sólo debemos aceptar como NIFs strings que tengan ocho números y una letra final. Comprobar que el Nif está en el formato correcto es responsabilidad de la función que transfiere NIFs. (Ver listado5).

En la función DDX_NIF() preparamos el control de edición para que se lleve el foco a éste si falla la transferencia. Mediante el miembro de CDataExchange *m_pDlgWnd* tenemos acceso al diálogo en el que tiene lugar la transferencia, con su función *GetDlgItem()* obtenemos un puntero al control de edición del NIF.

m_bSaveAndValidate nos permite conocer la dirección de la transferencia. Si estamos transfiriendo hacia variable, leemos el contenido del control con su función miembro *GetWindowText()*. A continuación la función ValidaNIF() comprueba si el formato de este texto es correcto. Si no lo es mostramos un mensaje de error y llamamos a Fail(), lo que interrumpe la transferencia de datos y devuelve el foco al control NIF.

*Las funciones DDV se utilizan habitualmente para validar datos*

Si estamos transfiriendo datos de variable a control, simplemente ponemos el texto de la variable en el control con su función miembro *SetWindowText()*.

## DDV Propios

Las funciones DDV se utilizan para validar datos, es decir, examinar si éstos cumplen determinadas condiciones y darlos por correctos o incorrectos. Si no cumplen alguna condición, la transferencia debe fallar, mostrando un mensaje que informe al usuario sobre el error y UpdateData() debe devolver FALSE , para que nuestro programa tenga conocimiento de que la transferencia ha fallado.

Estas validaciones se efectúan sobre los datos una vez han sido transferidos por el DDX correspondiente. Las comprobaciones que van ligadas al propio tipo de dato y que no dependen de ninguna condición deben incluirse en el propio DDX, por ejemplo, si una variable debe ser numérica y en el control correspondiente escribimos una letra, la transferencia debe fallar en el DDX. Por el contrario, si una vez transferido un número deseamos comprobar que este esté dentro de un rango determinado utilizaremos un DDV. (Ver listado 6).

**Listado 6**

```
void
AFXAPI DDV_LonCodigo(CDataExchange*
pDX, const CString& sCodigo, int nLon)
{
  if(pDX->m_bSaveAndValidate &&
            sCodigo.GetLegth()!=nLon) {
       pDX->m_pDlgWnd->
      MessageBox("código de longitud incorrecta", "Error",
      MB_ICONEXCLAMATION);
      pDX->m_pDlgWnd->MessageBox();
      pDX->Fail();
}
}
void
CSocioDlg   ::
   DoDataExchange(CDataExchange* pDX)
{
CDialog::DoDataExchange(pDX);
            ...
DDX_Text(pDX, IDC_CODIGO,
            m_Socio.m_sCodigo);
DDV_LonCodigo(pDX,      m_sCodigo,
m_nLonCodigo);
}
```

En el diálogo tenemos el control "código" que lee una clave alfanumérica que identifica al Socio. Para transferir este tipo de dato usamos una variable CString y el DDX_Text de las MFC. Deseamos que esta clave tenga siempre un número determinado de caracteres. Para comprobar si la clave introducida es de la longitud adecuada implementamos una función DDV_LonCodigo(). A ésta le pasamos como parámetros: un puntero a CDataExchange, una referencia al string que queremos validar y un entero que indica el número de caracteres que debe tener. DDV_LonCodigo realiza la comprobación únicamente si estamos transfiriendo hacia variables. Si el código no es de la longitud correcta, muestra un mensaje descriptivo y llama a Fail(). Esta función abortará la transferencia, haciendo que UpdateData devuelva FALSE y llevará el foco al ultimo control que haya sido preparado con PrepareControl(). El DDX_Text estándar de las MFC ya ha llamado a PrepareControl() para IDC_CODIGO, por lo que el foco será llevado al control correcto. Para que esto funcione, la llamada al DDV debe hacerse en la línea siguiente al DDX_Text. Como norma todos los DDX de las MFC empiezan llamando a PrepareControl() y los DDV llaman a Fail() sin preparar el control.

## Conclusión

En los ejemplos anteriores hemos analizado diversos casos en los que, más allá de la transferencia automática que proporciona ClassWizard, ha sido necesario un conocimiento profundo de los mecanismos de transferencia de las MFC. Un buen conocimiento de estos mecanismos nos permitirá superar, de forma elegante, numerosos problemas que se nos plantean habitualmente en la programación.

## Contactar con el autor

Xesco Hernández es colaborador de BIT (formación informática) y su dirección de e-mail es xesco@redestb.es.

LINUX

# Programación X-Window: Cursor Hardware

*Jorge F. Delgado Mendoza*

En anteriores artículos realizamos una implementación del driver de una tarjeta de vídeo SVGA para XFree86. Al dar soporte a modos SVGA conseguimos millones de colores en resolucionessuperiores a 1000 puntos por línea. Pero aún no hemos utilizado la característica más sobresaliente de una tarjeta de vídeo moderna, el acelerador gráfico.

En este artículo introduciremos al lector en el procedimiento para añadir rutinas aceleradas, añadiendo soporte para la más extendida de las aceleraciones: el *Cursor Hardware*. Tras el último artículo de la serie, nuestro trabajo parecía completo. Al fin y al cabo, ya podemos utilizar nuestra maravillosa tarjeta de vídeo *Accelerated'r'Us* con resoluciones de hasta 1280x1024 puntos en pantalla, así como con profundidades de hasta 24 bpp (*bits per pixel*) o lo que es lo mismo 16'7 millones de colores en pantalla.

Sin embargo, al soportar los modos SVGA disponibles, sólo hemos explotado un pequeño porcentaje de las capacidades de las Tarjetas de Vídeo modernas. Como ya describíamos en el primer artículo de la serie, la evolución de estos periféricos no se detuvo en la incorporación de modos de vídeo con mayor resolución (cantidad de puntos en pantalla), mayor profundidad (número de colores presentados simultáneamente) o ambas cosas a la vez.

Todas las tarjetas modernas poseen lo que se ha dado en llamar un *Acelerador de Interfaces Gráficas de Usuario* (*GUI Accelerator*). Este componente no es más que un microprocesador dedicado a realizar, de manera totalmente independiente de la CPU principal del sistema, las primitivas de dibujo que con más frecuencia se presentan en un entorno de ventanas. También se le conoce vulgarmente como acelerador 2D, ya que está especialmente diseñado para aumentar el rendimiento de las funciones gráficas utilizadas en la presentación de objetos de dos dimensiones. El número y calidad de las funciones que posee un acelerador varía con la calidad y el precio de la tarjeta, pero hoy en día suele incluir:

- **Expansión de Color:** en la mayoría de las aplicaciones gráficas, - edición de texto, hojas de cálculo, etc.., - es muy común que todo lo que se dibuje tenga solo dos colores: primer plano y fondo. El *expansor de color* permite preprogramar estos tonos, de tal manera que cada *bit* escrito en la memoria de vídeo representa un punto, en vez de una pequeña fracción del mismo. De esta manera multiplicamos la velocidad con la que trasladamos información desde la CPU a la tarjeta de vídeo, aumentando el ancho de banda por un factor de 8, 16 ó 24 en función del número de *bits per pixel* del modo que se esté utilizando.

- **BitBlt Engine:** una de las operaciones más comunes de un entorno gráfico es la utilización de primitivas que muevan partes de la pantalla de un lugar a otro. Estas funciones pueden estar debidas a movimientos de ventanas, *scrolling* de texto, etc. Además, es posible rellenar grandes superficies de un color o diseño determinado a base de dibujar una pequeña parte y trasladarla al resto del área de dibujo. Este componente

permite copiar o mover partes de una pantalla de un lugar a otro con unos pocos parámetros. *BitBlt Engine* significa literalmente "Motor de Transferencia de Bloques de Bits".

- **Dibujado de Líneas:** imprescindibles en la creación de formas complejas e importantes a la hora de presentar ventanas en la pantalla, las primitivas que dibujan líneas, - horizontales, verticales o generalizadas, - grupos de líneas o segmentos, constituyen uno de los cuellos de botella más importantes de las operaciones de dos dimensiones. El motivo es sencillo: se necesita un gran número de ciclos de reloj de la CPU para calcular la posición exacta de cada punto en la pantalla. Los aceleradores de líneas resuelven este problema calculando y dibujando toda la línea a partir de las posiciones de los extremos y un par de parámetros auxiliares.

- **Cursor Hardware:** probablemente la función más extendida y la más fácil de utilizar, motivos por los cuales nos va a servir de ejemplo para codificar la primera rutina acelerada del servidor. Su misión es actualizar el cursor automáticamente, sin necesidad de intervención por parte de la CPU.

Además del acelerador 2D, existen otros dos tipos de coprocesadores gráficos cuya presencia en el mercado es menos importante, principalmente debido a su reciente aparición y por la falta de un entorno que los haga imprescindibles. Estos aceleradores son:

- **Acelerador 3D:** diseñados para evitar a la CPU el trabajo de realizar las operaciones más comunes asociadas al cálculo y presentación de universos en tres dimensiones. Las funciones que suelen estar implementadas por *hardware* incluyen: Sombreado *Gouraud*, *Bilinear Filtering*, *Texture Mapping*, *Anti-aliasing*, *Z-buffering*, *Alpha Blending*, etc.

- **Acelerador de Vídeo:** aparecieron gracias al *boom* de la Multimedia, para paliar las deficiencias de los PCs a la hora de reproducir vídeo de alta calidad con un número razonable de imágenes por segundo. En general suelen proporcionar: Escalado de Imágenes, Descompresión de MPEG, *Overlay*, etc.

A lo largo de este artículo vamos a aprender a *enganchar* una función acelerada al *driver* de un servidor de *X-Window*. Para ello vamos a utilizar la más sencilla de todas: el *cursor hardware*.

A pesar de no ser la función más fácil y rápida de implementar, su especial interés didáctico se debe principalmente a dos factores:

- **Claridad Funcional:** el código necesario para proporcionar al servidor los servicios de un *cursor hardware* está estructurado en bloques claramente diferenciados, separados en función del servicio que proporcionan. Gracias a ello, es posible realizar una aproximación incremental al problema, añadiendo funciones gradualmente. Las ventajas de este método son evidentes: depuración más sencilla, fácil comprensión de lo que se está realizando, etc.

- **Correlación Hardware Software:** en la mayoría de los aceleradores *hardware* las funciones implementadas no se corresponden exactamente con las primitivas de dibujo del servidor de ventanas. Es, por tanto, trabajo del desarrollador adaptar aquellas rutinas del acelerador que considere apropiadas para que realicen operaciones útiles para el sistema de ventanas, lo que complica el trabajo de diseño de las rutinas de aceleración. En el caso del *cursor hardware*, no es necesario hacer la adaptación, ya que la función para la que está diseñado es exactamente la que realizan las primitivas *software* del servidor, pudiendo reemplazarlas directamente.

Estos, y no otros, son los motivos por el cual utilizaremos el *cursor hardware* como primera función acelerada a implementar. En próximos artículos aprenderemos a adaptar herramientas del acelerador de tal manera que sirvan para nuestros propósitos.

Antes de comenzar con la lista de requisitos, el aviso de siempre: nunca se ha de llevar el *hardware* más allá de sus especificaciones. Una Tarjeta de Vídeo quemada o un tubo de rayos catódicos fundido es la recompensa habitual del desarrollador inconsciente.

# Requisitos

Como ya es normal en la serie de artículos, lo primero que vamos a hacer es establecer los requisitos, materiales y de conocimientos, necesarios para que el lector pueda seguir el proceso de desarrollo e implementar una solución práctica para su propia Tarjeta de Vídeo.

### • Conocimientos

Como ya viene siendo normal en toda la serie de artículos, es necesario dominar el lenguaje de programación C; herramienta imprescindible a la hora de incorporar nuevas funciones al servidor.

Unas nociones de ensamblador del *x86*, especialmente del manejo de puertos de entrada/salida es más que aconsejable, ya que todas las interacciones realizadas con el acelerador se hacen a través de *I/O Ports*.

Finalmente, es necesario tener una idea bastante clara de la estructura interna de los mapas de memoria, tanto del PC como del *framebuffer* de la Tarjeta de Vídeo.

### • Software

Para poder jugar con las capacidades avanzadas de una tarjeta, vamos a necesitar los siguientes paquetes *software*:

- *gcc 2.7.0* o superior.
- *make*.
- *XFree86*.
- *LinkKit*.

- **Documentación**

El documento principal que necesitamos es el manual de registros de la tarjeta, en el que se describa con detalle la interfaz del *cursor hardware*. Un manual de Lenguaje C, así como las especificaciones de la Tarjeta y el monitor serían muy recomendables.

Útil, pero no imprescindible, es una descripción de los interfaces de la tarjeta de vídeo con sus distintos componentes. Si además conocemos el ancho de banda de los distintos interfaces, podremos tomar decisiones más acertadas a la hora de implementar rutinas de aceleración *hardware*.

Finalmente, podremos encontrar información muy interesante, - aunque en inglés, - en el documento titulado *Definition of the Porting Layer for the Xv11 Sample Server*. Este documento se puede encontrar en el árbol de fuentes del servidor del *X-Window System* como *ddx.ps*.

# Cursor Hardware: Qué Es y Para Qué Sirve

A la hora de decidir qué herramientas *hardware* se deben integrar en un coprocesador gráfico, los diseñadores de Tarjetas de Vídeo utilizan dos criterios que aunque a primera vista pudieran parecer contrarios son, en realidad, complementarios:

- **Frecuencia de Utilización:** se analizan en profundidad las *Interfaces Gráficas de Usuario*, buscando cuellos de botella en el proceso de presentación de información. Como resultado de este análisis se ha descubierto que de todas las primitivas gráficas existentes, sólo un número muy reducido de ellas se utilizan de manera frecuente, mientras que las demás aparecen muy rara vez.

- **Facilidad de Integración:** el otro gran factor analizado es comúnmente conocido como *relación calidad/precio*, es decir, qué funciones gráficas es posible implementar en un *microprocesador dedicado*, de tal manera que el coste del mismo sea lo más bajo posible.

*Todas las interfaces gráficas utilizan un cursor en todo momento y su implementación resulta muy sencilla*

Como resultado de este proceso, de las primitivas más comunes en los entornos gráficos, - rellenado, líneas, cursor, transferencias, etc., - se eligen para su implementación aquellas cuya integración en un microprocesador sea eficiente desde el punto de vista *complejidad/rendimiento*. Finalmente, se añaden aquellas funciones menos comunes que sean baratas de implementar, - bien porque son pequeñas modificaciones de las ya realizadas o bien porque su incorporación es fácil y directa.

El caso del *cursor hardware* es muy especial. Por un lado, su implementación es muy sencilla, ya que sólo requiere unos cuantos registros y, en la mayoría de los casos, un acceso extra al DAC de la Tarjeta de Vídeo. Por el otro, todas las interfaces gráficas utilizan un cursor en todo momento.

Ambos motivos hacen que el *cursor hardware* esté presente en prácticamente todos los aceleradores gráficos de hoy en día.

# Funcionamiento de un Cursor Software

Para entender las ventajas que proporciona un *cursor hardware*, vamos a estudiar qué pasos se siguen habitualmente en el desarrollo de las *Interfaces Gráficas de Usuario* a la hora de presentar en la pantalla la dichosa flechita apuntadora.

Desde el punto de vista de la codificación *software* existen dos alternativas de diseño que son claramente opuestas:

- La primera de ellas da prioridad al cursor sobre las demás operaciones, realizándose los cálculos necesarios para evitar tapar el cursor al ejecutar las primitivas gráficas. La imagen del cursor se actualiza únicamente si éste se desplaza o cambia de aspecto.

- En la segunda alternativa, la prioridad del cursor es la más baja de todas. Las demás primitivas se ejecutan olvidándose de la existencia del *sprite* del cursor, el cual se actualiza al final de cada conjunto de operaciones básicas.

Ambas alternativas tienen sus ventajas e inconvenientes. La complejidad de cálculo de la primera de ellas puede estar compensada por la cantidad de actualizaciones que se han de realizar en la segunda.

En la práctica se suele utilizar un sistema híbrido entre ambas, si bien tiene preferencia la segunda de las opciones comentadas.

El enorme ancho de banda de las interfaces de las tarjetas que se utilizan habitualmente hoy en día, provocan que el coste temporal de transferir una pequeña imagen a la pantalla sea muy bajo si lo cmparamos con el complejo cálculo de fronteras que es necesario para *esquivar* el cursor.

# ¿TE ATREVES?

Desde **Sólo Programadores** estamos preparándonos para un torneo muy especial, de ordenador a ordenador, de programador a programador. El torneo de ajedrez está empezando a tomar forma y éstas son las bases, aunque también podéis informaros en AWS en la calle Serrano Jover, 3, de Madrid, o en el teléfono (91) 5 42 50 87. Para conocer cuál será el alcance de la participación, os pedimos que os inscribáis enviando vuestros datos a la dirección de AWS o a la dirección de e-mail: **solop@towercom.es**. El periodo de inscripción ya ha comenzado y terminará el 30 de noviembre, y la fecha límite para la recepción de los programas es el 15 de diciembre de 1997. La celebración del torneo será anunciada con suficiente antelación y podréis participar en persona o desde vuestra propia ciudad, ya que vuestro programa os representará y en todo momento estará presente un notario.

## BASES DEL TORNEO

1. Se presentará un programa que juegue al ajedrez recibiendo la jugada del contrario y devolviendo la propia. El tiempo de la partida será medido por un juez y la partida se jugará físicamente en una mesa con un tablero siguiendo las instrucciones que cada ordenador dé; cada ordenador avisará de que ha elegido un movimiento con un pitido y mostrará su jugada en la pantalla de la forma: casilla a mover - casilla destino. La notación de la casilla vendrá dada por su columna y su fila. Para las columnas se utilizarán las de la A a la H siendo la columna A la primera columna de la izquierda, visto el tablero desde la posición de las blancas. Para las filas utilizaremos los números del 1 al 8 siendo la fila 1 la más cercana a nosotros (visto el tablero desde el lado de las blancas). Como ejemplo, el movimiento del peón de rey dos lugares hacia adelante sería: "E2 - E4". La jugada que hace el contrario se anotará en el ordenador de forma manual tecleando "E2-E4" <RETURN>.
2. El programa jugará al ajedrez según las normas de este juego, admitiendo todos sus movimientos.
3. El programa se ejecutará bajo Windows 95, MS Dos 6.22 y Linux, en un Pentium 166 con 16 MB de RAM.
4. El programa podrá jugar con las blancas o con las negras indistintamente.
5. Se debe entregar el código fuente, la versión del compilador utilizado y una breve documentación para su utilización.
6. Un programa podrá ser descalificado si realiza un movimiento no válido y esto deberá ser advertido por el contrario; por lo tanto, un movimiento será válido siempre que el otro programa no nos advierta de lo contrario. De la misma forma, si un programa nos advierte de un movimiento erróneo y éste no lo es, será descalificado.
7. Los jueces son los encargados de advertir las posiciones de tablas, el programa no advertirá de ello.
8. El programa debe detectar si ha ganado la partida y quedará descalificado ante cualquier detección errónea. No se avisará al contrario en el caso de que deje al rey al descubierto y ganará la partida el que antes mate al rey.
9. La metodología del concurso se reservará hasta que sea conocido el número de participantes (eliminatoria, puntos, etc.)
10. No se podrán utilizar en el programa bases de datos adicionales.
11. Los derechos del programa ganador pasarán a la propiedad de la empresa Tower Communications, SRL.
12. El premio consistirá en un Pentium PRO 200 con 64 MB RAM, VIRGE4 MB, CDX 16, disco duro 5 GB, monitor 17", teclado, ratón y disquetera.

# CONCURSO DE PROGRAMACIÓN

SÓLO PROGRAMADORES Y AWS INFORMATICA

SERRANO JOVER 3, MADRID

# Funcionamiento del Cursor Hardware

El comportamiento del *cursor hardware* es radicalmente diferente al descrito para el caso no acelerado. El objetivo es minimizar los tres parámetros que más tiempo de CPU consumen:

- Utilización del *bus*.
- Repintado del cursor.
- Cálculo de fronteras.

Para ello, el *bitmap* (imagen) asociado al cursor se guarda en la memoria de la Tarjeta de Vídeo, en una posición que no interfiera con el modo de vídeo que se esté utilizando. Si estamos utilizando un modo de 1024x768 en 8 bpp, el cual necesita 768 Kbytes, y tenemos 2 Mbytes instalados en la tarjeta, podremos colocar el *bitmap* del cursor en cualquier posición a partir del Kbyte 769. Una serie de registros informan al acelerador de la posición exacta de la imagen del cursor en el *frame buffer.*

Un segundo grupo de registros se encarga de informar al acelerador de las coordenadas lógicas *x* e *y*, referidas a la pantalla de vídeo, donde tiene que aparecer el cursor. Mediante una modificación del contenido de los registros de posición, el cursor se mueve automáticamente a las nuevas coordenadas.

Finalmente, un último grupo de registros se encarga de controlar las diferentes características del cursor, como pueden ser: parpadeo, colores de primer plano y fondo, velocidad del parpadeo, etc.

Gracias a este sistema, el servidor puede dibujar en la pantalla olvidándose del cursor ya que, como se puede ver en la Figura 1, la pantalla y el cursor se tratan como dos fuentes de vídeo independientes. Normalmente es el *hardware* asociado al DAC el encargado de mezclar ambos flujos de vídeo.

Mediante la utilización del *cursor hardware* se consigue cumplir los objetivos que hemos expuesto anteriormente:

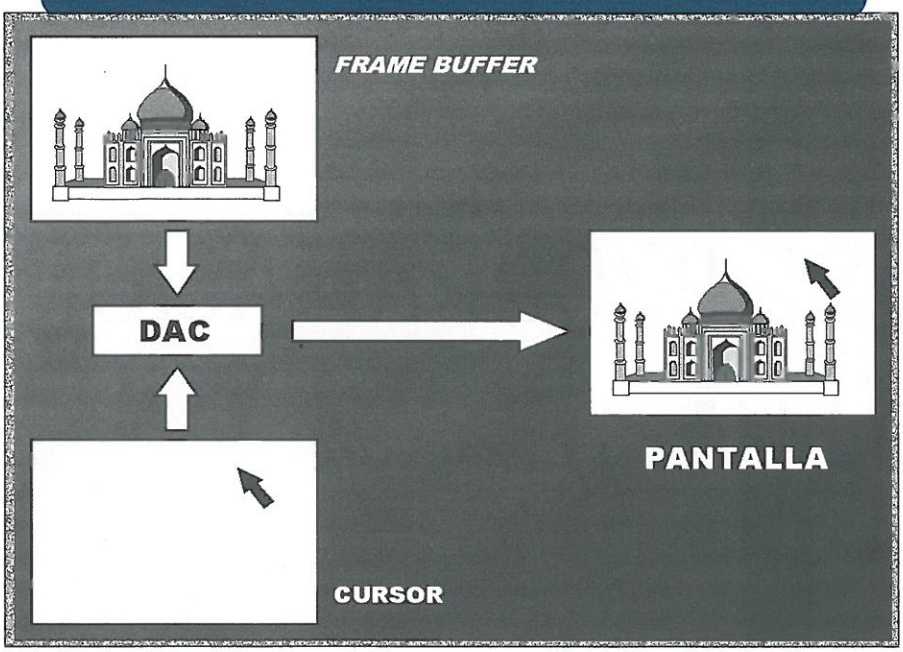
- **Utilización del bus:** el *bitmap* del cursor sólo se envía a la tarjeta de vídeo cada vez que cambia el dibujo del mismo. De esta manera evitamos todas las transferencias debidas a cambios de la imagen del cursor. Del mismo modo, cada vez que se mueve el cursor transmitimos dos coordenadas en lugar de toda la imagen, reduciendo al mínimo el uso de la interfaz.

*El bitmap del cursor sólo se envía a la tarjeta de vídeo cada vez que cambia su dibujo*

- **Repintado del Cursor:** no es necesario repintar el cursor cada vez que se realiza alguna operación en la pantalla, ya que el acelerador se encarga de combinar ambas imágenes. Como consecuencia se evita el incómodo parpadeo del cursor debido normalmente a una actualización rápida de los contenidos de la pantalla.

- **Cálculo de Fronteras:** al ser tratados como flujos de vídeo independientes, desaparece el problema de calcular los solapamientos de las imágenes y el cursor. Del mismo modo, desaparece la necesidad de guardar las imágenes que oculta el cursor, así como la de actualizar el fondo de la pantalla cada vez que se mueve el apuntador.

Figura 1. En una implementación típica de cursor hardware, la imagen de la GUI y la del cursor proceden de dos fuentes distintas, siendo el RAMDAC el encargado de mezclarlas. Este método elimina complicados cálculos de borde así como las operaciones de actualización/repintado de la pantalla cada vez que se mueve el cursor.




# Una Nueva Estructura: Cada Cosa en su Sitio

Hasta ahora hemos utilizado un único fichero para la codificación en C del servidor, añadiendo más y más código según hemos ido añadiendo funcionalidad a

Tabla 1. Ficheros que componen el *driver* de las tarjetas de *Sólo Programadores Tech. Inc.*

| Nombre | Contenido |
|---|---|
| gu_driver.h | Prototipos de Funciones, Definiciones Comunes y Macros. |
| gu_driver.c | Código del driver SVGA, Enganches de las Funciones Aceleradas. |
| gu_cursor.c | Código Relativo al Cursor Hardware. |
| Imakefile | Relación de Ficheros a Compilar. |

nuestro *driver*. Sin duda es la manera más sencilla, pero a partir del millar de líneas de código empezamos a tener una situación inmanejable a la hora de mantener el *software*.

Para corregir esta situación vamos a reestructurar el código, moviendo todas las partes comunes a un fichero único y utilizando un archivos distintos para almacenar el código que proporcione funcionalidades diferentes.

No existe una única manera de realizar esta distribución; nosotros vamos a utilizar una que consideramos lógica, creando la estructura de ficheros que se describe en la Tabla 1. De esta manera podremos tener cada componente principal en su propio fichero, pudiendo realizar modificaciones cómodamente.

La nueva estructura del *driver* nos va a forzar, además, a cambiar el *Imakefile*, de tal manera que se compilen todos los ficheros y se enlacen de manera adecuada.

Una vez realizados todos los cambios, realizaremos el código relativo al cursor en el fichero **gu_cursor.c**, tocando el código principal del *driver* para incluir las rutinas aceleradas en la función *FbInit()* del fichero **gu_driver.c**.

Una última advertencia: después de repartir el código en varios ficheros conviene ver si el *driver* funciona correctamente *antes* de empezar a añadir código relativo al *cursor hardware*. De esta manera aislaremos los errores que se hayan podido producir al migrar a la nueva estructura, antes de confundirlos con los fallos que introduzcamos con el código del cursor.

## Implementación del Cursor Hardware

Desde el punto de vista del servidor, el cursor es una estructura con tres componentes fundamentales:

- La imagen o *bitmap* del cursor.
- Un grupo de funciones concebidas como puntos de acceso al cursor y que se encargan de realizar todas las operaciones relativas a él: mover el cursor, ocultarlo, etc.
- Datos extras del cursor: parpadeo, tamaño, etc.

Para utilizar el cursor del acelerador *hardware*, deberemos reemplazar las funciones *software* del servidor por otras que, manteniendo la equivalencia funcional, manipulen los diferentes parámetros del acelerador.

Para ello reemplazaremos casi todas las funciones de la estructura asociada al cursor con las que desarrollemos. El lugar donde debemos realizar esta sustitución es la función *FbInit()* del servidor, donde ya dijimos que se realizaban todas las inicializaciones relacionadas con la aceleración *hardware*.

De ahora en adelante, el servidor utilizará el *cursor hardware* por defecto, por lo que añadiremos al servidor la opción: **Option "sw_cursor"**. Si encontramos esta línea en el fichero de configuración **/etc/XF86Config**, deberemos inhibir el *cursor hardware*, utilizando las rutinas propias del servidor de *X-Window*.

## Enlace con el Servidor: FbInit()

En esta función del *driver*, vamos a realizar una serie de comprobaciones que determinarán si podemos activar el *cursor hardware* o no. Para ello analizaremos las siguientes características del modo de vídeo a través de la estructura *DisplayModeRec*:

- *Profundidad:* el *cursor hardware* de nuestra tarjeta de vídeo *Accelerated'r'Us* sólo funciona en modos de 256 colores. Si al iniciar el modo, el número de colores es distinto de 256, utilizaremos el cursor *software* original del servidor del *X-Window System*.
- *Memoria Libre:* como se puede deducir de una atenta lectura del manual de la Tarjeta de Vídeo, el gráfico correspondiente al *cursor hardware* se guarda en la memoria de vídeo o *frame buffer*. El espacio necesario para almacenar el cursor es de 8 Kbytes, - cuatro de ellos para los colores de primer plano y fondo mientras que los otros cuatro son para la máscara. Esta característica del cursor nos fuerza a deshabilitarlo siempre que no quede memoria libre disponible en la tarjeta.
- *Opciones:* finalmente, vamos a desactivar el *cursor hardware* si se nos indica en el fichero de configuración mediante la opción:

  **Option "sw_cursor"**

Una vez hayamos determinado que el *cursor hardware* se encuentra disponible, sustituiremos las funciones apropiadas del servidor por sus versiones aceleradas

**Figura 2. Registros asociados al Cursor Hardware. Los índices 0x20 a 0x23 marcan la posición del cursor relativa a las coordenadas de la pantalla.**

**3DF** **HC MSB Horizontal** **Índice = 20** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Bits más significativos de la posición horizontal del cursor, relativo al comienzo de la pantalla, en puntos. La esquina superior izquierda es el origen de coordenadas. |

**3DF** **HC LSB Horizontal** **Índice = 21** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Bits menos significativos de la posición horizontal del cursor, relativo al comienzo de la pantalla, en puntos. La esquina superior izquierda es el origen de coordenadas. |

**3DF** **HC MSB Vertical** **Índice = 22** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Bits más significativos de la posición vertical del cursor, relativo al comienzo de la pantalla, en puntos. La esquina superior izquierda es el origen de coordenadas. |

**3DF** **HC LSB Vertical** **Índice = 23** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Bits menos significativos de la posición vertical del cursor, relativo al comienzo de la pantalla, en puntos. La esquina superior izquierda es el origen de coordenadas. |

**Figura 3. Registros asociados al Cursor Hardware. Los índices 0x24 y 0x25 marcan el tamaño del sprite. Los registros 0x26 a 0x28 deben contener la posición del gráfico del cursor en la memoria de vídeo.**

**3DF** **HC Comienzo Horizontal** **Índice = 24** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Estos bits definen la posición horizontal de comienzo del cursor dentro de la cuadrícula de 64x64 puntos. El cursor siempre acaba en la posición 63. |

**3DF** **HC Comienzo Vertical** **Índice = 25** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Estos bits definen la posición vertical de comienzo del cursor dentro de la cuadrícula de 64x64 puntos. El cursor siempre acaba en la posición 63. |

**3DF** **HC Posición en Memoria I** **Índice = 26** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Bits más significativos de los 24 bits que marcan el comienzo del *buffer* de 64x64 puntos en la memoria de vídeo. |

**3DF** **HC Posición en Memoria II** **Índice = 27** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Bits intermedios de la palabra de 24 bits que determina la posición de comienzo del *buffer* de 64x64 píxels en la memoria de vídeo. |

basadas en el *hardware* de la tarjeta. Además, inicializaremos las estructuras de datos relativas al cursor, de manera que sean coherentes con el estado del mismo.

Para el diseño del código, se ha utilizado como modelo el que se emplea en el *driver* de las tarjetas *Cirrus Logic 592x*. Para otros modelos es posible que algunas funciones sean innecesarias o bien se necesiten algunas que no están reflejadas en el artículo.

## Función CursorInit()

La primera de las funciones necesarias a la hora de implementar el *cursor hardware* es la de inicialización. Éste es el lugar indicado para:

- Inicializar los parámetros relativos al cursor que pertenecen a las estructuras de datos del servidor del *X-Window System*.
- Dar los valores por defecto del cursor.
- Rellenar todos aquellos parámetros de la tarjeta que se sepan de antemano: posición del *sprite* en la memoria de vídeo, parpadeo del cursor, posición inicial en la pantalla, etc.

## Función ShowCursor()

Su única utilidad es activar el cursor físicamente. Para ello accederemos a los registros relativos al *cursor hardware* de la tarjeta, colocando los valores necesarios para que el cursor quede activado.

Para activar el cursor y no sobreescribir los valores que se introdujeron en la función de inicialización, realizaremos las siguientes operaciones:

- Accederemos al puerto 0x3DE para seleccionar el registro 0x2B (*Hc* Control) a través del puerto 0x3DF.
- Guardaremos el valor que tenga el registro en una variable temporal.
- Pondremos a '1' el *bit* '0' de la copia temporal del registro para así activar el cursor una vez se introduzca el valor a través de 0x3DF.
- Volcaremos el contenido de la variable temporal en 0x3DF.

## Función HideCursor()

De manera similar a la función anterior, *HideCursor* se encarga de desactivar el cursor físicamente en la tarjeta. El modo de hacerlo es idéntico, utilizando una variable temporal y siguiendo los pasos:

- Colocaremos en el registro índice el valor necesario para acceder al registro de control del cursor.

**Figura 4. Registros asociados al Cursor Hardware. Los índices 0x29 y siguientes se encargan de controlar los diferentes parámetros del cursor hardware.**

**3DF** **HC Posición en Memoria III** **Índice = 28** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Bits menos significativos de la palabra de 24 bits que determina la posición de comienzo del cursor *hardware* en la memoria de vídeo. |

**3DF** **HC Color 0** **Índice = 29** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Color 0 del *sprite* del cursor. |

**3DF** **HC Color 1** **Índice = 2A** **R/W**

| Bit | Descripción |
|---|---|
| 7-0 | Color 1 del *sprite* del cursor. |

**3DF** **HC Control** **Índice = 2B** **R/W**

| Bit | Descripción |
|---|---|
| 0 | HC Control |

| Bit 0 | Descripción |
|---|---|
| 0 | Cursor desactivado. |
| 1 | Cursor activado. |

| Bit | Descripción |
|---|---|
| 1 | Parpadeo. |

| Bit 1 | Descripción |
|---|---|
| 0 | Parpadeo desactivado. |
| 1 | Parpadeo activado. |

| Bit | Descripción |
|---|---|
| 3-2 | Control del Parpadeo. |

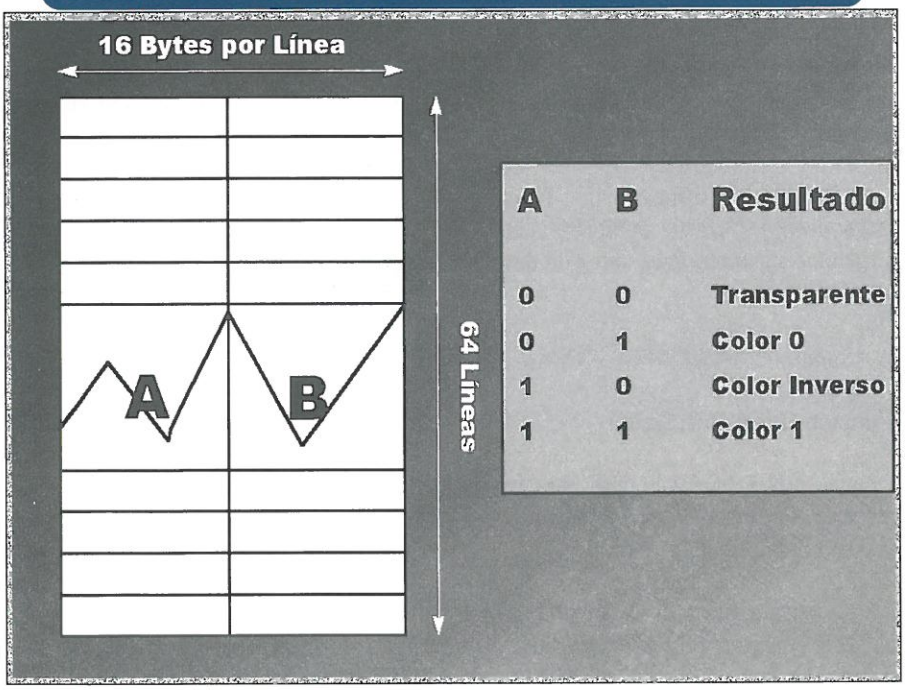
| Bit 3-1 | Descripción |
|---|---|
| 0 0 | 4 refrescos activado y 4 desactivado. |
| 0 1 | 8 " " 8 " |
| 1 0 | 16 " " 16 " |
| 1 1 | 32 " " 32 " |

| Bit | Descripción |
|---|---|
| 7-4 | Reservados. |

**Por Defecto: 0x0.**

Nota: El Cursor Hardware solo funciona en modos de 256 colores.

**Figura 5. Cada punto del cursor ocupa dos bits en la memoria de vídeo. El primero de ellos (A) selecciona el color, - fondo o primer plano, - en función del contenido de los registros 0x29 y 0x2A. El segundo bit (B) es la máscara que determina si un punto determinado se verá en la pantalla o no.**




- Guardaremos el contenido en la variable temporal.
- Modificaremos la variable adecuadamente, colocando en el *bit* '0' el valor '0'.
- Volcaremos el contenido en el registro de control a través de 0x3DF.

### Función LoadCursorToCard()

Se utiliza para cargar un nuevo *sprite* de cursor en la memoria de la Tarjeta de Vídeo. En el código ejemplo vamos a utilizar una única posición para guardar el gráfico del cursor, por lo tanto cada vez que éste cambie, deberemos almacenar los datos relativos al nuevo *bitmap*.

### Función LoadCursor()

Aunque puede parecer redundante al existir *LoadCursorToCard()*, esta función es necesaria y es una de las más importantes, ya que realiza todas las manipulaciones necesarias de las estructuras de datos del servidor y la tarjeta, para cargar una imagen nueva y actualizar su estado.

Las operaciones que se han de realizar son las siguientes:

- Se comprueba que los parámetros del servidor son correctos.
- Se desactiva el cursor utilizando *HideCursor()*.
- Se carga el nuevo curso en la memoria de vídeo con *LoadCursorToCard()*.
- Se actualiza la paleta de colores vía *RecolorCursor()*.
- Actualizamos la posición del cursor en la pantalla de vídeo con *MoveCursor()*.
- Volvemos a activar el cursor utilizando la función *ShowCursor()*.

Una vez realizados estos pasos, el nuevo cursor aparecerá en la pantalla sin

que se aprecien parpadeos o ruido en la transición.

*En la función FbInit() del servidor, se realizan todas las inicializaciones relacionadas con la aceleración hardware*

**Función SetCursor()**

Es uno de los puntos de entrada del código del servidor a la hora de interactuar con el cursor. Esta rutina se limita a actualizar el *punto caliente* del cursor en la estructura interna del mismo y llamar a *LoadCursor()*.

**Función RestoreCursor()**

Otro de los puntos de entrada, esta función es muy similar a la anterior, aunque su misión es restaurar todos los datos relativos al cursor. Para ello utiliza una función general, independiente de la máquina, llamada *miPointer Position()*.

Una vez nos devuelvan el control, de nuevo invocaremos *LoadCursor()* para actualizar los valores internos al controlador *hardware* del cursor.

**Función MoveCursor()**

Uno de los puntos de acceso más críticos desde el punto de vista del rendimiento al ser probablemente el más utilizado. Se encarga de mover el *cursor hardware* en la pantalla mediante la inserción de los valores apropiados en los registros de control del acelerador.

El procedimiento a seguir es el siguiente:

- Actualizamos las coordenadas del cursor en función del *punto caliente* del mismo.
- Comprobamos que el cursor está dentro de la pantalla, en caso contrario preparamos el parpadeo del cursor.
- Si las coordenadas del *sprite* se salen de la pantalla, las colocamos en el extremo de la misma.
- Actualizamos los registros cuyos índices van del 0x20 al 0x23 para reflejar el cambio de coordenadas del cursor.

**Función RecolorCursor()**

Se encarga de transportar los colores de primer plano y fondo a la paleta utilizada actualmente por el modo de vídeo. Si tuviéramos un *cursor hardware* que operara en modos de diferente profundidad de color, deberíamos codificar esta función.

Al poderse utilizar el cursor en modos de 256 colores nada más, utilizaremos una función independiente de máquina que ya está implementada en el servidor, de nombre:

*miRecolorCursor()*

**Función WarpCursor()**

De nuevo una función cuya implementación ya está realizada en el código de la capa *DIX* (*Device Independent X*) del servidor.

Podremos utilizar la función *miPointerWarpCursor()* siempre y cuando añadamos siguiente línea antes de devolver el control:

*xf86Info.currentScreen = pScr;*

**Función QueryBestSize()**

Se utiliza habitualmente para interrogar al *hardware* acerca del tamaño ideal o recomendado del cursor.

Normalmente utilizaremos tamaños de 32x32 puntos para el cursor del *X-Window System*.

## Conclusiones

A lo largo de este artículo, el tercero de la serie dedicada a *Linux* y *X-Window*, hemos aprendido la forma de incorporar funciones, que hagan uso del acelerador de la Tarjeta de Vídeo, al *driver* del servidor de *X*. Para ello, hemos incluido soporte para la más sencilla, desde un punto de vista didáctico, de las capacidades de nuestra tarjeta imaginaria *Accelerated'r'Us*: el *cursor hardware*.

También hemos aprendido los criterios que se siguen a la hora de diseñar y construir un acelerador, ya sea de 2D, 3D o vídeo. Finalmente, hemos profundizado en nuestros conocimientos relativos a la arquitectura interna de una Tarjeta de Vídeo moderna.

*QueryBestSize() se utiliza para interrogar al hardware sobre el tamaño ideal del cursor*

En el próximo artículo de la serie, añadiremos funciones que utilicen el *expansor de color* de la tarjeta. Gracias a él podremos incluir soporte para rellenado rápido, una de las aceleraciones más espectaculares desde el punto de vista del usuario.

## Contactar con el Autor

Jorge Delgado Mendoza es miembro de XFree86 desde el año 1994, como desarrollador responsable del servidor de Oak Technologies Inc. Es Ingeniero Superior de Telecomunicaciones por la UPM y trabaja actualmente en Convex Supercomputers. Su dirección de correo electrónico es, provisionalmente, ernar@dit.upm.es.

# Aprendizaje Automático y Vida Artificial

INTELIGENCIA ARTIFICIAL

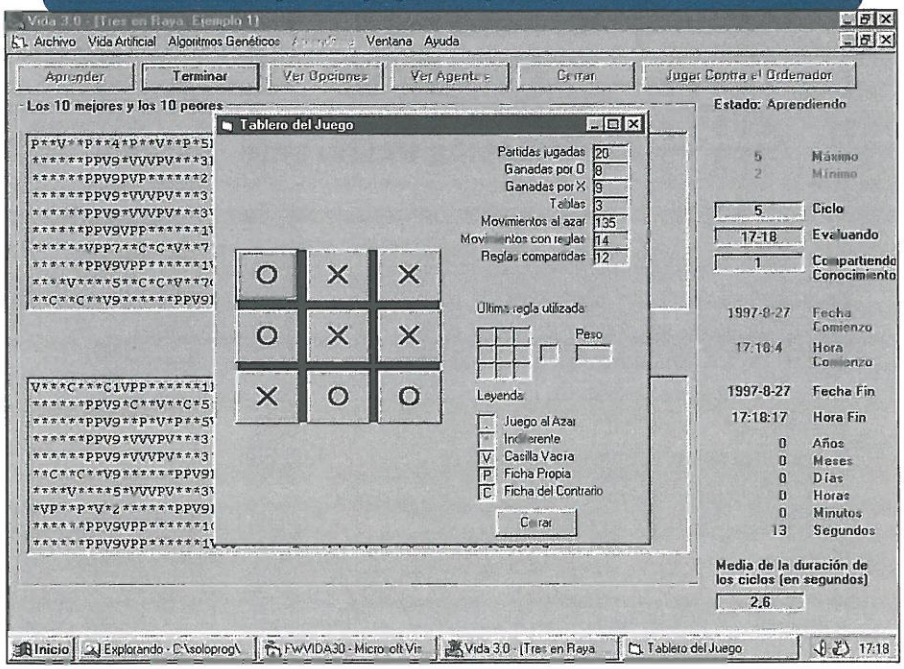
*Manuel de la Herrán Gascón*
*mherran@usa.net*

Seguimos intentando crear vida. Esta vez, vida inteligente. Este artículo describe cómo utilizar los conceptos de Vida Artificial y Algoritmos Genéticos para crear programas capaces de aprender por sí mismos a resolver todo tipo de problemas. Como ejemplo se ofrece en el CD-ROM el código fuente de un programa capaz de aprender a jugar al tres en raya.

En anteriores números de la revista se ha estudiado el tema de la Vida Artificial (*Sólo Programadores*, nº 36) con un programa que permitía observar fenómenos emergentes como la evolución y la cooperación en un mundo virtual habitado por hormigas y plantas. También se han descrito los Algoritmos Genéticos (*Sólo Programadores*, nº 37): qué son, cómo programarlos y cómo mejorarlos. Su funcionamiento se puso a prueba en un programa que trataba de descubrir una frase introducida por el usuario. Se va a recordar brevemente este Algoritmo Genético (AG). En primer lugar se genera una lista de frases combinando palabras tomadas al azar de un diccionario. Las frases que más se parecen a la frase buscada se seleccionan como "buenas" y se "reproducen" por parejas, creando nuevas frases por combinación de palabras tomadas de uno y otro progenitor. Repitiendo el proceso de selección y reproducción, y permitiendo de vez en cuando la modificación de una palabra al azar (mutación), se llega a obtener la frase buscada.

En un AG, los elementos que se reproducen (en este ejemplo, frases) son llamados entidades, agentes, cadenas,

Figura 1. El programa en pleno aprendizaje.



patrones, soluciones o simplemente "bichos". Uno de los problemas que surgen a la hora de utilizar algoritmos genéticos, y que se discute en el número nº 37 de la revista, es saber cuáles son las entidades que deben ser seleccionadas. En el caso de las palabras y las frases, se usa la propia solución (la frase buscada) para saber hasta que punto una frase es correcta, por lo que se puede decir que el programa, en realidad no adivina nada, ya que sabe por anticipado exactamente qué es lo que está buscando. Sin embargo, el ejemplo fue muy útil para experimentar mejoras en el rendimiento del algoritmo.

Esta vez tenemos la oportunidad de comprobar cómo este método se puede aplicar a un problema en el cual no tenemos por qué conocer los elementos que componen el agente buscado, aunque sí sabemos que es lo que queremos que haga: jugar y ganar al tres en raya. Se trata de un juego muy simple y no es difícil programar un ordenador para que lo haga con cierta habilidad. Lo que hace interesante este programa es que no requiere ningún conocimiento previo sobre el juego. Es decir, el programa será capaz de jugar contra sí mismo y contra nosotros, y aprender a partir de la propia experimentación, y poco importa que se trate del tres en raya o del ajedrez, ya que no va a ser necesario introducir al programa conocimiento sobre cómo jugar, y mucho menos sobre cómo ganar.

## El programa debe aprender a jugar por sí mismo, sin ninguna ayuda

En el CD-ROM que acompaña a esta revista se encuentra el programa de instalación para Windows 3.x, 95 y NT, y el código fuente en Visual Basic 4.0 para 16 bits de *Ejemplos de Vida 3.0*, que incluye el mundo artificial de hormigas y plantas, el algoritmo genético capaz de adivinar frases y el programa que aprende a jugar al tres en raya.

## Aprendiendo a jugar al Tres en Raya

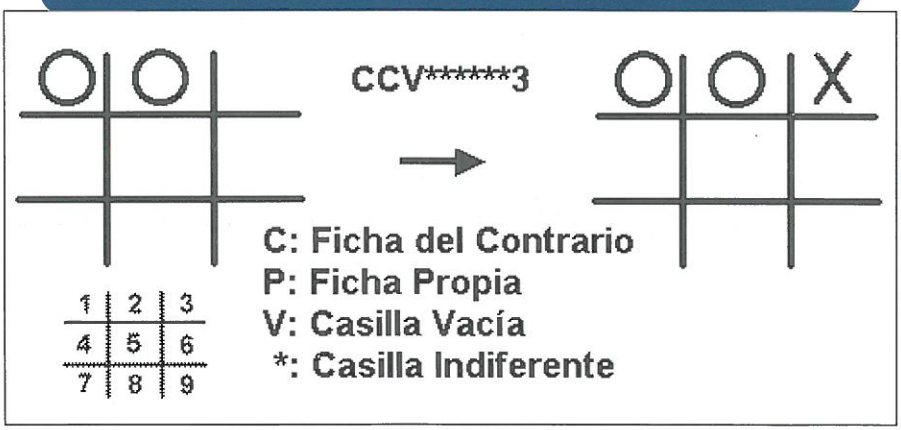
En el programa que adivina frases, los bichos son frases y están compuestos de palabras que se combinan en la reproducción. En el tres en raya, los bichos son "jugadores", y están formados por "reglas" que definen cómo actuar (jugar) ante determinadas circunstancias. El programa maneja reglas del tipo *"Si observo tal situación, realizo tal acción"* (Ver figura 1). Al jugar una partida, la entidad observa el estado del tablero. Si posee alguna regla que aplicar, la aplica. En caso contrario, juega al azar.

el agente seleccionará la regla que más victorias haya proporcionado. Para ello, al igual que los agentes, las reglas también tendrán asignado un *peso*, que aumentará cada vez que la regla haya sido usada en una partida ganada y disminuirá en las perdidas.

Estas reglas son el *material genético* que maneja el programa. Cuando dos jugadores se reproducen, generan una nueva entidad formada por las reglas de ambos tomadas de forma alternativa, y creando de vez en cuando una nueva regla al azar en vez de tomarla de uno de los padres.

Básicamente, el proceso de aprendizaje consiste en evaluar las entidades, ordenarlas en función de su utilidad y reproducir los mejores individuos, que ocuparán el lugar de los perdedores. Evaluar las entidades en este caso es agruparlas por parejas y hacer que jueguen partidas.

Figura 2. Esta regla evita que el contrario ponga tres fichas en línea.




Todos los jugadores poseen un indicador llamado *peso* que representa la habilidad de dicho agente en el juego del tres en raya. Cuando finaliza una partida entre dos jugadores virtuales, el peso del ganador se incrementará en una unidad, al contrario que el del perdedor. En el caso de que la partida finalice en tablas, sin ganadores, los pesos no se verán afectados.

Si en un momento dado de la partida la entidad puede aplicar más de una regla,

```
Sub bucle_general()
  While hay_que_detener = False
    evaluar_agentes
    ordenar_agentes
    reproducir_agentes
  Wend
End Sub
```

Cuando consideremos que el proceso de aprendizaje ha durado lo suficiente (y esto puede ser mucho tiempo), pulsaremos el botón de "Terminar" y después el de "Jugar contra el Ordenador" para

comprobar cuánto se ha aprendido. Pulsando "Ver Agentes" veremos una lista de las reglas de cada agente, con los pesos finales de cada regla y de cada agente.

Al jugar contra el programa, en realidad estamos jugando contra el primer agente de la lista, es decir, el que más veces ha ganado, aunque el programa permite también jugar contra cualquiera de los otros que han quedado "vivos". En el tablero de juego se informa, entre otras cosas, del número de partidas que han finalizado en tablas y del número de fichas totales que se han colocado utilizando una regla (en vez de al azar), por cada ciclo.

## Trampas e Inevitables

La premisa que se ha seguido en todo el proceso de programación ha sido la de no incluir en el programa ningún conocimiento acerca del problema a resolver, ya que lo que se está estudiando es el aprendizaje en general, y no la representación del conocimiento humano para un problema en particular.

Sin embargo, en algunos aspectos es inevitable incluir cierto conocimiento y en otros se ha hecho "trampa". Los aspectos inevitables son, por ejemplo, los que se refieren al juego en sí mismo, como el formato de las reglas. Cada regla posee en la parte izquierda nueve valores que corresponden con los nueve sentidos del agente, y en la parte derecha un valor que representa una de las nueve acciones posibles.

Evidentemente las entidades deben ser capaces de ver el tablero y de colocar fichas, y esto también hay que programarlo. Otro aspecto inevitable es la función que detecta cuál de los jugadores ha ganado la partida, o si el juego ha quedado en tablas. Es decir, el juego, el problema a resolver o el entorno donde habita el agente (como se quiera llamar), hay que programarlo: no podemos resolver un problema si no definimos los términos de ese problema.

## Los agentes ven el tablero como una secuencia de datos

Por otra parte, el programa produce un aprendizaje muy lento. Recordemos que el programa no selecciona jugadores buenos al tres en raya (ya que no sabe lo que es un jugador bueno, precisamente eso es lo que tratamos de descubrir), sino que se seleccionan jugadores *mejores que los demás*. El mejor individuo de todos estará muy preparado para jugar y ganar en **su** mundo. Bueno, pero un niño descubre casi inmediatamente cómo no perder en el juego. ¿Por qué nuestro programa es tan torpe? No hay que olvidar que los agentes ven el tablero como una secuencia de datos, es decir, tal como aparece en la figura 3.

Figura 3. Así ve el agente el tablero.

Hay que reconocer que incluso para nosotros resultaría muy incómodo jugar con un tablero así. Si además no supiésemos que esta fila de casillas representa en realidad una rejilla de 3x3, podríamos volvernos locos intentando averiguar cómo ganar una partida.

Estas razones animan a cometer algunas excepciones en la generación de reglas para acelerar el proceso del aprendizaje. En primer lugar, siempre se generan reglas válidas. Esto no tendría por qué ser así, podría por ejemplo generarse la regla C1 que no es válida ya que coloca la ficha en una casilla ocupada. Los agentes que aplicasen una regla de este tipo perderían la partida inmediatamente, y así sólo se conseguiría retardar aún más el aprendizaje. El programa también permite dar mayor probabilidad de aparición a cierto tipo de reglas que se sabe de antemano que pueden ser muy interesantes, concretamente reglas con dos P ó dos C y una V, que en muchos casos podrán corresponder con: "Si tengo dos fichas en línea pongo la tercera y gano" o "Si el contrario tiene dos fichas en línea, le impido ganar poniendo la tercera".

## Y el agente... ¿Qué piensa de todo esto?

Este programa se ha realizado, por comodidad, desde el punto de vista del *proceso*. Existen unos *operadores* (como el cruce y la mutación) que son aplicados en serie a todas las entidades. Es decir, primero se evalúan todas las entidades, después se seleccionan las mejores, más tarde se reproducen, etc. Cada agente decide cómo jugar al tres en raya, pero no puede decidir con quién jugar, con quién reproducirse, o cuándo compartir su conocimiento.

## Es posible programar estas simulaciones desde el punto de vista de las entidades

Para dotar a los agentes de mayor autonomía, a la vez que se logra una aproximación más cercana al mundo real, es preferible programar estas simulaciones desde el punto de vista de las entidades

(los datos), siendo la propia entidad la que decide en cada momento cómo actuar.

Los dos estilos de programación pretenden simular un paralelismo: el primero es un paralelismo de procesos, el segundo es un paralelismo de entidades. El ideal sería disponer un procesador para cada entidad (paralelismo de entidades real). A falta de ello, el programa podría repartir el tiempo de ejecución entre todas las entidades, y el bucle general quedaría de la forma:

```
Sub bucle_general()
 While hay_que_detener = False
  for i=1 to numero_de_seres_vivos
     ejecutar_ser_vivo
   next i
 Wend
End Sub
```

Esto recuerda mucho más al bucle general del programa de hormigas y plantas que al de las palabras y frases. Es más complejo de programar por el hecho de tener que mantener el estado en que se ha quedado cada entidad antes de pasar a la ejecución de la siguiente, pero tiene la ventaja de que gran parte del código sería independiente del problema que se está resolviendo.

¿Cómo se consigue el mismo resultado dejando la iniciativa a las entidades? Bueno, en realidad los agentes no van a tener demasiada libertad, aunque tal vez ellos piensen lo contrario. En primer lugar es recomendable añadir a las entidades unas coordenadas que hagan referencia a su ubicación física, permitiendo que se muevan al azar por su mundo virtual. A partir de ahora, dos entidades deberán estar juntas para poder reproducirse o jugar una partida. Esto sería lo equivalente a la función de desordenar que ya posee el programa. Por otro lado, llamemos al peso "energía".

La energía aumenta al ganar partidas y disminuye al perderlas. Hagamos que sólo se puedan reproducir los agentes con un determinado nivel de energía, y que fallezcan aquellos en los que ésta ha llegado por debajo de cierto valor. Acciones como reproducirse, moverse o la simple inactividad pueden restar energía al agente. Como se puede suponer, los agentes no van a tener más remedio que jugar y ganar si quieren seguir estando "vivos" o tener descendencia.

La energía global del sistema debería permanecer constante (o al menos no variar demasiado) por coherencia, y para evitar extinciones o superpoblación, que de todas formas podrían suceder, y que en general se han de evitar.

## La reproducción se simplifica mucho, al ser las propias entidades las que eligen pareja

Este modelo tiene muchas ventajas. Por una parte, una vez construido el esquema general, nos damos cuenta de que resulta mucho más fácil y rápido programar algunas funciones. La reproducción se simplifica mucho, al ser las propias entidades las que eligen pareja. Además este modelo nos permite ampliar y enriquecer el concepto de agente proporcionándole mayor potencia, convirtiéndolo en un pequeño resolutor de problemas. Y no sólo eso: para solucionar un problema distinto, bastará con definir unos nuevos agentes; el resto del programa permanecerá igual.

Vamos a ver poco a poco todas las cosas que nuestros agentes son capaces de hacer. Cada entidad posee una serie de *acciones* que es capaz de realizar, pero ahora no sólo coloca fichas: ahora también puede, por sí misma, compartir su conocimiento, buscar pareja, reproducirse, o comenzar una partida. Podemos hacer que en cada ciclo, cada entidad sólo pueda realizar un número determinado de acciones, elegidas del conjunto de acciones posibles, antes de pasar a la ejecución de la siguiente (recordemos que hay que simular un paralelismo). Otra posibilidad es asignar un intervalo de tiempo a cada entidad y que cada una ejecute todas las acciones que le sea posible. También podremos limitar el tiempo de ejecución (o el número de acciones) a algunas de las entidades peores, en vez de eliminarlas.

En el programa de las hormigas, cada entidad realizaba una acción por ciclo, que dependía del estado en que se encontraba, calculado en función de los vecinos que tuviera en ese momento. En el caso de que un mismo estado pudiera disparar más de una acción, la acción se decidía según una cierta probabilidad. Ahora estamos hablando de algo mucho más completo, más sencillo de entender (ya que se parece más a la vida real), aunque más tedioso de programar. La entidad poseerá básicamente cuatro atributos: objetivo, creencias, sentidos y acciones. El agente (autómata) se dedicará a una u otra tarea (acción, salida) en función de un proceso de decisión interno (inteligencia), más o menos complejo, basado en sus creencias (memoria, reglas) y sentidos (entradas).

## La entidad poseerá cuatro atributos: objetivo, creencias, sentidos y acciones

La entidad va a tener un objetivo que cumplir y sus sentidos le informarán, entre otras cosas, de cuándo ese objetivo se cumple y en qué grado. Utilizando sus creencias, el agente realiza en cada caso la acción que cree que más le acercará a su meta. En el caso del tres en raya, el objetivo es ganar el juego. Los sentidos informan en cada momento del estado del tablero, que se almacenará en la memoria. Las acciones son no sólo las que actúan directamente sobre el problema que se está resolviendo, cómo colocar una ficha, sino también las que afectan a cualquier

otro componente del entorno (cómo compartir el conocimiento con otro agente), o aquellas que consisten en manejar de alguna forma el conocimiento almacenado en la memoria, y que posteriormente determinarán la secuencia de acciones a realizar sobre el entorno, como predecir el comportamiento del contrario.

## Un estado es un conjunto de valores para las variables que definen el entorno

La información que recibe el programa acerca del entorno se almacenará en variables. Un *estado* será un conjunto de valores para esas variables. Por ejemplo, nuestro agente podrá ver el contenido de cada una de las 9 casillas del tres en raya, con lo que dispondrá de 9 variables, y las posibles combinaciones de esos valores formarán el conjunto de estados posibles.

## Creencias

Aquello en lo que la que la entidad cree lo podemos agrupar en tres categorías. En primer lugar, existirá una memoria de los sucesos que han ocurrido. Por otro lado es posible que unas variables dependan del valor de otras, por lo que existirán reglas que definan esta descomposición. Finalmente tenemos las reglas que predicen el comportamiento del entorno. Las reglas usadas por el juego del tres en raya son del tipo *"si observo tal situación, realizo tal acción"*. Estas reglas sirven para decidir qué hacer, pero no explican el porqué. Un formato más general debería permitir reglas capaces de describir un estado inicial, un estado final, y la acción necesaria para pasar de uno a otro (Ver figura 4). Reglas como (1) *"Si es el turno de X, yo juego con X, y en las casillas 1 y 2 hay una ficha X, estando vacía la casilla 3, y realizo la acción de poner una ficha en la posición 3, entonces el ganador será X con una probabilidad del 100%, y esta regla tiene una importancia máxima"*. Podrían existir reglas sin acción y/o reglas donde el estado final se deba calcular en función de otras variables: *"Si en las casillas 1 y 2 hay una ficha X, en las casillas 4 y 5 hay una ficha O, estando vacías las casillas 3 y 6, entonces el ganador será el del turno actual, con una probabilidad del 40%, y esta regla tiene una importancia de 0,7"*.

Un formato que admite estas posibilidades es:

Estado, Acción —> Conclusión, Probabilidad, Importancia

*Estado* es un conjunto de pares "variable = valor", *Acción* es una acción simple o compleja, *Conclusión* es un conjunto de pares "variable = fórmula", que una vez resueltas las fórmulas produce pares "variable = formula". *Probabilidad* es la confianza que el agente tiene en la regla, e *Importancia* es un indicador del grado en que la regla puede afectar al objetivo.

Figura 4. Algunas posibles reglas.

| | ESTADO | ACCION | | CONCLUSION | PROB. | IMP. |
|---|---|---|---|---|---|---|
| (1) | Turno Actual = X<br>Yo Juego Con = X<br>P1 = X<br>P2 = X<br>P3 = V | 3 | → | Ganador = X | 100% | 1 |
| (2) | P1 = X<br>P2 = X<br>P3 = V<br>P4 = O<br>P5 = O<br>P6 = V | 0 | → | Ganador = Turno Actual | 40% | 0.6 |

## Acciones

¿Cómo decide la entidad qué acción ejecutar con este tipo de reglas? La entidad posee una variable especial, llamada *objetivo*, que representa el grado en el que la entidad se comporta según la forma deseada por quien la creó. La variable objetivo dependerá de los valores de otras variables, según una fórmula determinada, en este caso, la colocación de tres fichas propias en línea.

La evaluación de la entidad, o lo que es lo mismo, la comprobación de hasta dónde se cumple el objetivo, está encapsulado dentro de la propia entidad. No es necesario que otro agente lo evalúe, la propia entidad no puede evitar perder "energía" si no lo consigue.

El agente intentará realizar aquellas acciones que más le acerquen a su objetivo. Para ello, deberá predecir la respuesta del entorno ante sus acciones. Pero recordemos que el agente parte de un completo desconocimiento del problema particular. ¿Qué hacer? En primer lugar, observa la situación actual (inicialmente, el tablero vacío), y comprueba si el objetivo buscado se cumple.

Como todavía nadie ha sido declarado ganador, y a falta de otra opción mejor, el agente ejecuta acciones al azar. Después de una acción observa sus consecuencias y su "mente" produce una asociación causa-efecto que almacena en forma de regla. Las posteriores acciones estarán condicionadas por el conocimiento almacenado en la memoria, evitándose en lo posible recurrir al azar.

Si se realiza una acción y se pasa a un estado desconocido, se almacenará información que represente ese nuevo

# suscríbete

## a Sólo Programadores

## y consigue un magnífico descuento

**suscripción normal**

ahorro **20%**

12 revistas (1 año) por sólo...

**9.350** ptas.

**suscripción estudiantes**
(carreras técnicas)

ahorro **40%**

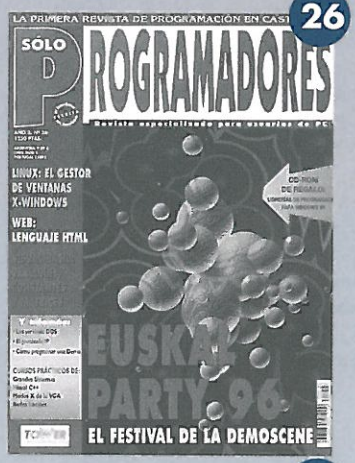
12 revistas (1 año) por sólo...

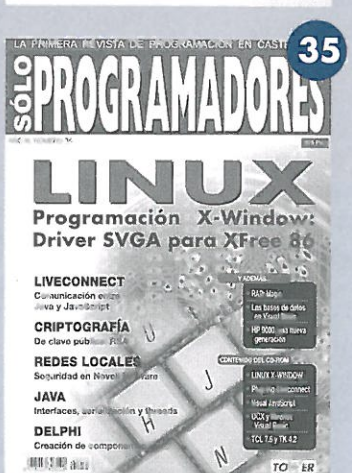
**7.050** ptas.

# números atrasados

# SÓLO PROGRAMADORES

# completa ya tu colección





estado y se creará una regla que represente las condiciones necesarias para pasar a ese nuevo estado. El problema consiste en cómo identificar qué acción ha sido la responsable de una transición. Por ejemplo, colocar una ficha en un determinado lugar puede ser la causa inmediata de ganar la partida, pero tiene poca utilidad suponer que únicamente esa acción es la responsable de vencer en el juego. Es más apropiado decir que la causa es la sucesión completa de acciones desde el comienzo de la partida.

Para poder utilizar este tipo de reglas, las acciones se podrán organizar en una estructura jerárquica. Así, una sucesión de acciones interesante formará una acción de nivel superior, y esta acción podrá formar parte de una regla.

## El problema consiste en cómo identificar qué acción ha sido la responsable de una transición

Sin embargo esto no explica la mayoría de los comportamientos inteligentes conocidos. Una partida no se gana por realizar una determinada serie de acciones, sino más bien por una estrategia, por un conocimiento del juego en general, por un conocimiento de experto obtenido a través de múltiples experiencias. A este nivel de análisis, la causa de haber ganado es el haber realizado una serie de acciones -tomar una decisión es realizar una acción-, que además de actuar sobre el entorno o problema a resolver, manejan símbolos referentes a él y manipulan o indican cómo manipular la información adquirida, determinando en última instancia cuál será la sucesión de acciones a realizar sobre el juego.

Aunque en cada instante sólo se pueda ejecutar una acción básica, las acciones de nivel superior normalmente se

### Fragmento de código del programa que se adjunta en el CD-ROM

```
Function f_funcion_evaluacion_agentes_3r() As Integer

    'Devuelve 1 si ha ganado el primero, es decir O
    'Devuelve 2 si ha ganado el segundo, es decir X
    'Devuelve 0 si es tablas

    'Ahora juegan una partida los dos agentes:


    Dim cont_agente As Integer
    Dim cont_regla As Integer
    Dim azar As Integer
    Dim i As Integer
    Dim peso As Double
    Dim ganador As String * 1

    'el primer jugador juega con O          (numero_de_agente)
    'el segundo jugador juega con X         (numero_de_agente + 1)
    'el jugador que cominenza se elige al azar

    'Elegir un jugador de comienzo al azar
    azar = Fi_Azar1(2)
    If azar = 1 Then
       turno_3r = "X"
    Else
       turno_3r = "O"
    End If

    'inicializamos el tablero
    For i = 1 To numero_de_caracteres_parte_izq_regla_3r
       estado_del_tablero_3r(i) = "V"
       lista_de_casillas_libres_3r(i) = i
       If ver_agentes_3r Then
          frm_juego3r.b(i).Caption = ""
       End If
    Next i
    numero_casillas_libres_3r = numero_de_caracteres_parte_izq_regla_3r

    ganador = "N"
    While ganador = "N"
       s_es_el_turno_de
       ganador = f_ganador_3r()
       s_cambio_de_turno_3r
    Wend

    'el primer jugador juega con O          (numero_de_agente)
    'el segundo jugador juega con X         (numero_de_agente + 1)
    'el jugador que cominenza se elige al azar
    If ganador = "O" Then
       f_funcion_evaluacion_agentes_3r = 1
    Else
       If ganador = "X" Then
          f_funcion_evaluacion_agentes_3r = 2
       Else
          'ganador es "T"
          f_funcion_evaluacion_agentes_3r = 0
       End If
    End If
```

pueden solapar en el tiempo. La toma de una decisión cuya influencia se manifiesta en el desarrollo de otras acciones puede entenderse también como una acción.

Por ejemplo, el hecho de decidir utilizar, bajo ciertas condiciones, únicamente un subconjunto del conjunto total de reglas posibles, puede considerarse una acción (de nivel superior). Esta acción se solapará con las acciones básicas que se ejecuten en cada momento. Este ejemplo además ilustra cómo permitir una cierta jerarquía dentro de las reglas, ya que es posible incluir esta acción, que hace referencia a las reglas, dentro de otra regla.

## Una partida no se gana por realizar una serie de acciones, sino más bien por una estrategia

En las variables también existe una jerarquía. El número de variables que forman un estado normalmente será muy grande, y muchas veces ocurrirá que varios estados distintos se comportan como uno sólo. Esta última situación se da en el tres en raya por la simetría que posee el tablero. Por estas razones es interesante representar algunas combinaciones de valores de variables como un único valor para una variable de nivel superior.

Así, dispondremos de un mecanismo que nos permitirá algo parecido a crear *conceptos*. Por ejemplo, se podría crear el concepto de "dos fichas en línea", que será una nueva variable, con un nombre o identificador cualquiera, que adoptará el valor verdadero en el caso de que se produzca esta situación. En las variables es posible aplicar la *lógica difusa*. Existen conceptos de naturaleza continua, que se definen mediante la referencia a una propiedad y el grado en que se manifiesta dicha propiedad. Por ejemplo, nosotros podemos decir que el cielo es azul. Sin embargo, no existe una frontera numérica que marque la diferencia entre un cielo azul y un cielo blanco en función del número de puntos de luz de cada color. Ambos conceptos pueden ser ciertos simultáneamente en cierta medida. Así, podríamos expresar: el cielo posee un tono azulado en un 70%. Por otra parte no siempre se dará un grado de confianza total a la información manejada. Es posible que un concepto se cumpla en un cierto grado y que a esta información se le asigne una cierta confianza, y de esta forma, podríamos decir: el cielo posee un tono azulado en un 70% y esta información posee una probabilidad del 90% de ser cierta. Estos valores pueden ser incluidos en cada variable.

¿Cómo reconocer que conceptos son importantes? Cuando un agente llegue a su límite de almacenamiento, deberá seleccionar algunas creencias para que sean borradas. Para ello, intentaremos relacionar los estados con el objetivo, de manera que se tengan más en cuenta aquellos que más influyen en él, ya sea favoreciendo o perjudicando su consecución.

## El tiempo se puede considerar como una variable

La creación de variables de nivel superior también hace que las necesidades de espacio a la hora de almacenar una regla sean menores. Con un número suficiente de reglas describiendo el comportamiento del entorno, y conociendo el estado actual y el estado al que se desea llegar, se puede buscar la sucesión de acciones que nos llevan de un estado a otro, hasta la meta. La creación de variables superiores no ha de estar limitada a un solo instante de tiempo, a una sola lectura del entorno. En realidad, el tiempo se puede considerar como una variable más, y crear así conceptos que describan un comportamiento que posea continuidad.

## Programación Orientada a... ¿Agentes?

En este artículo se han descrito las posibilidades de razonamiento dentro de un agente. Las entidades que hemos estudiado recuerdan a los procesos Unix (que nacen, mueren, tienen procesos hijos, se comunican enviándose mensajes), y también tienen un aire de instancias de *Programación Orientada a Objetos* (POO), ya que si en POO tenemos objetos que encapsulan datos y procedimientos, aquí tenemos agentes que encapsulan un objetivo, unos sentidos, unas creencias y unas acciones.

En cualquier caso, lo que es evidente es que si el método de búsqueda es independiente del problema a resolver, se echa terriblemente en falta la existencia de unas librerías de entidades y de un compilador de un lenguaje de definición de agentes y de entornos de problema. En teoría, con algo así, nos quedaría poco por programar. En la práctica, y dada la lentitud inherente del aprendizaje evolutivo, cuyo ejemplo más cercano somos nosotros mismos (que hemos tardado 3.800 millones de años en evolucionar desde los primeros microorganismos vivos), será más útil combinar estas técnicas con la representación humana del conocimiento. De todo ello puede resultar un *Resolutor General de Problemas* de gran potencia. Las posibilidades de este nuevo paradigma son sin duda, muy excitantes.

En el próximo número veremos los agentes desde una perspectiva más elevada: nos centraremos en los mundos donde viven y en cómo crear mundos o universos paralelos llenos de agentes y sacar provecho de todo esto.

PROGRAMACIÓN EN JAVA

# El sistema de Entrada/Salida

El modo de dar una vez en el clavo, es dar cien veces en la herradura.

Miguel de Unamuno

Cualquier programador podría hacer suya la frase de Unamuno, pues en 16 palabras, define el arte/ingeniería de la programación. Los creadores del lenguaje Java, no debían tener muchas ganas de golpear con el martillo, cuando diseñaron el sistema de entrada/salida de nuestro Java. Ya que nos encontramos con una abigarrada colección de clases, derivadas de *InputStream* y *OutputStream*, con nombres, en demasiadas ocasiones, poco intuitivos, y cuya única manera (efectiva) de utilizar, es mediante mezclas y pastiches, como en una coctelera (y todos sabemos que la creación de un buen cóctel, no es asunto baladí). Esta situación podía haber cambiado con la versión 1.1, pero no fue así (siempre nos quedará París...) y con la excusa de la internacionalización, añadieron un "emplasto" en forma de dos jerarquías más, *Reader* y *Writer*.

De acuerdo, utilizo estas nuevas clases y me olvido de las antiguas, que además dan unos feos mensajes en tiempo de compilación, pensará el lector. Pero, en ciertas ocasiones, estaremos obligados a emplear las clases "viejas", en combinación con las "nuevas". Por ejemplo, si deseamos trabajar con archivos comprimidos.

Esperando aliviar las jaquecas, erupciones, diarreas mentales, fiebres y demás efectos secundarios, que suele producir la primera cita a ciegas con este proceloso API, este artículo intentará ser una especie de Bálsamo de Fierabrás. Espero que no gastéis mucho dinero en aspirinas.

## Sobrevolando el bosque

Antes de comenzar a ver código, es interesante conocer con un cierto detalle (a vista de pájaro), la jerarquía de clases que nos permite leer y escribir datos en *streams*. De momento, nos concentramos en las clases "antiguas" (o *deprecated features,* como gusta llamarlas el compilador*)*.

Un *stream* puede verse como una secuencia ordenada de bytes, que manan desde una fuente y fluyen hacia un destino. Los recursos productores y consumidores de *streams*, pueden ser muy variados. Entre ellos se pueden citar: memoria, ficheros, un proceso ejecutándose en otro *thread*, Internet, etc.

En la librería de clases de Java para el manejo de *streams*, se establece la siguiente dicotomía, entrada y salida; esta dictadura bifronte es mantenida en el poder gracias a las clases abstractas *InputStream* y *OutputStream*. Por herencia, todas las clases derivadas de *InputStream*, contendrán métodos básicos tipo *read(),* para leer un simple byte o *array* de bytes. De igual forma, todas las clases derivadas de *OutputStream*, soportarán métodos tipo *write(),* para escribir un byte o un *array* de bytes.

Pero en el manejo de los *streams*, raramente utilizaremos una simple clase, lo

# ¿NECESITAS

## LA MEJOR REVISTA DE INFORMÁTICA

## INFORMARTE,

## LA MEJOR REVISTA PARA USUARIOS DE INTERNET

## NAVEGAR,

## LA MEJOR REVISTA DE JUEGOS

## JUGAR,

## LA MEJOR REVISTA DE PROGRAMACIÓN

## PROGRAMAR?

TOWER COMMUNICATIONS
C/ Aragoneses, 7 28108 Alcobendas (Madrid)
Tlfno. (91) 661.42.11 Fax. (91) 661.43.86

más normal será encadenar varias, hasta obtener la funcionalidad buscada.

## *Un stream no es más que un flujo de bytes entre dos puntos*

El siguiente *dueto* de clases básicas, lo constituyen las clases abstractas *FilterInputStream* y *FilterOutputStream*. Estas clases *Filter stream,* son subclases de *InputStream* y *OutputStream,* y no añaden nada nuevo a sus progenitoras. Las clases derivadas de estos filtros se limitan a realizar algún tipo de procesamiento con los bytes leídos o escritos por otro *stream*. Es decir, cuando creamos algún objeto tipo *Filter stream*, debemos pasarle una instancia de *InputStream* o *OutputStream* a su constructor.

Un par de *interfaces* que debemos conocer son *DataInput* y *DataOutput.* Los cuales, declaran métodos que nos permitan la lectura y la escritura de los tipos de datos básicos existentes en Java. Las siguientes clases los implementan:

- *DataInputStream* y *DataOutput Stream,* son clases filtro que ligaremos a un cierto *stream*, para poder leer/escribir los tipos primitivos, y los objetos *String* de una manera portable, entre distintas plataformas.

- *RandomAccesFile,* esta clase implementa las dos *interfaces*. Se encuentra fuera de las jerarquías *InputStream/OutputStream*, y nos permite abrir un fichero tipo sólo lectura, o también para lectura/escritura, ofreciéndonos la posibilidad de acceder a cualquier posición dentro del fichero (acceso aleatorio).

Fuera de las jerarquías comentadas se encuentra la clase *File*. Bajo este nombre, que puede parecer tan poco afortunado, los diseñadores del lenguaje decidieron encapsular las operaciones básicas asociadas a la gestión de un sistema de ficheros. Así podemos, entre otras cosas, listar y crear directorios, borrar o cambiar el nombre de ficheros, y obtener distintas informaciones acerca de un objeto *File.* Hay que tener bien claro que un objeto *File* se puede estar refiriendo tanto a un fichero como a un directorio. En la figura 1 podemos observar cómo encajan todas estas clases vistas, en la jerarquía de entrada/salida de Java.

## Tipos de Inputstream

En la tabla 1 podemos observar las diferentes subclases de *InputStream*. La tabla 2 pretende mostrarnos los diferentes filtros con los que contamos para mejorar el tratamiento de los *streams* de entrada. Estudiemos a continuación algunos ejemplos ilustrativos de las principales clases comentadas hasta el momento.

Figura 1.

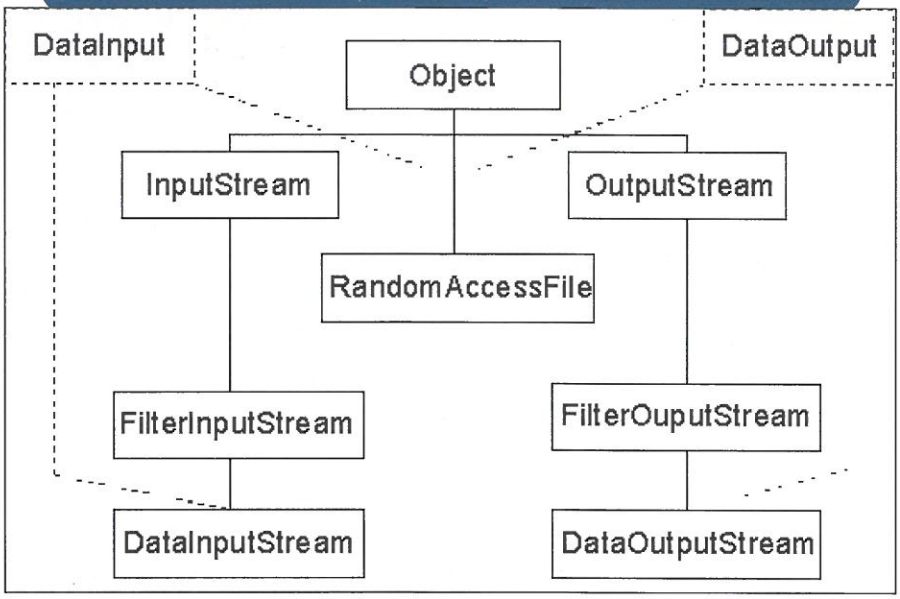



Tabla 1. Subclases de InputStream.

| Clase | Función |
|---|---|
| ByteArrayInputStream | Permite que un buffer en memoria sea usado como un *stream* de entrada. |
| StringBufferInputStream | Convierte una cadena (*String*) en un *stream* de entrada. |
| FileInputStream | Para leer información desde un fichero. |
| PipedInputStream | Produce los datos que serán escritos en un asociado *PipedOutputStream*. Utilizado para la transmisión de datos entre distintos *threads*. |
| SequenceInputStream | Convierte dos o más objetos *InputStream* en un simple *InputStream.* |
| FilterInputStream | Clase abstracta que sirve como base de las clases filtro, que añaden un cierto procesamiento a la simple lectura de bytes producida por las anteriores clases |

**Tabla 2 Filtros para InputStream**

| Clase | Función |
|---|---|
| DataInputStream | Utilizada en conjunción con *DataOutputStream*, para poder leer los tipos primitivos con los que cuenta el lenguaje. |
| BufferedInputStream | Usado para prevenir un lectura física, cada vez que necesitamos más datos. Es decir, implementamos un *buffer*. |
| LineNumberInputStream | Sigue la pista al número de líneas del *stream* de entrada. |
| PushbackInputStream | Mantiene un *buffer* gracias al cual podemos leer un byte del *stream*, pudiendo, posteriormente, retornarlo al *stream*, marcado como no leído. |

## ByteArrayInput Stream

El siguiente listado nos muestra la noción de marcado (*bookmark*), que soportan ciertos *streams* de entrada. Gracias al método *markSupported()*, comprobamos si el *stream* implementa dicha característica. El método *mark()*, marca la posición actual, para que podamos regresar a ella más tarde (mediante el método *reset()* ), debiendo especificar en sus argumentos, el número de bytes que podrán ser leídos antes de que la marca sea desactivada.

```
import java.io.ByteArrayInputStream;

class BAISExamp{
  public static void main(String[] args) {
   byte[] bArr = { 'a','b','c','d','e','f'};
   ByteArrayInputStream in = new
                ByteArrayInputStream(bArr);
    // available retorna (count - pos)
   // count <> Número total de crts. válidos
    // pos <> índice del próximo crt. que será leído
   System.out.println("Available: "+
                    in.available());
   int b;
   while ((b=in.read()) >= 0) // lee "abcdef"
     System.out.print((char)b);
   in.reset(); //pos=0
   System.out.println();
   in.skip(3);   // Nos saltamos "abc"
   if(in.markSupported()){
      in.mark(bArr.length-3);
      in.reset();//pos=3
   }
   System.out.println("Available: " +
                    in.available());
   while ((b=in.read()) >= 0)   // lee "def"
        System.out.print((char)b);
   System.out.println();
  }
}
```

*Las clases filtro refinan el procesamiento de los streams*

## StringBufferInput Stream

En este ejemplo hay que subrayar un pequeño detalle. Aunque utilizamos la clase filtro *DataInputStream* en conjunción con *StringBufferInputString*, a la hora de leer los caracteres de la cadena, utilizamos *readByte()* aplicando posteriormente un *casting* al tipo char, en vez de utilizar *readChar()*. Esta aparente incongruencia, se debe a lo mal que soporta la antigua jerarquía de entrada/salida los caracteres Unicode (que son de 16 bits), en este caso sólo los ocho bits más bajos de cada carácter de la cadena son utilizados por la clase. Cuando leemos del fichero, que se encuentra ligado a los filtros *BufferedInputStream* y *DataInputStream*, estamos leyendo bytes (*readLine()*, internamente, les aplica un *casting* a char), así que la cadena obtenida sólo contiene caracteres (16 bits) con valores ASCII (ocho bits). Posteriormente, cuando realizamos una lectura de *StringBufferInputString*, únicamente son tomados los ocho bits más bajos de cada carácter de la cadena (los correspondientes al código ASCII).

```
// SBISEx.java
import java.io.*;

public class SBISEx{
  public static void main(String[] args)
  throws IOException{
       // Varias envolturas para leer un fichero
       DataInputStream in =
        // Para poder leer líneas
        new DataInputStream(
            // Inmplementamos un buffer
           new BufferedInputStream(
             // Para trabajar con ficheros
            new FileInputStream("SBISEx.java")));
            String s, s2= new String();
            while ((s=in.readLine()) != null)
            s2 += s + "\n";
            in.close(); // Cierre del stream
       // Utilizamos como fuente de bytes
       // una cadena
    DataInputStream in2 =
        // Para poder leer tipos básicos
        new DataInputStream(
             new StringBufferInputStream(s2));
        while(in2.available() != 0)
          System.out.print((char)in2.readByte());
  }
}
```

## PipedInputStream PipedOutputStream

Estas dos clases cimentan el concepto de tubería (*pipe*) de comunicaciones. *PipedInputStream* representa el final de la tubería, hacia donde fluye la información generada por la clase *PipedOutputStream*. Suelen ser utilizadas para intercambiar datos entre dos procesos. Por ejemplo, un *thread* enviará

datos a través de una instancia de *PipedOutputStream*, y el otro *thread* leerá dichos datos a través de una instancia de *PipedInputStream*. Veamos una pequeña muestra para aclarar el funcionamiento:

```
// PipeTest.java

import java.io.*;

// Lee bytes de un stream de entrada
class ByteReader extends Thread{
 private DataInputStream in;

 public ByteReader(InputStream i)
 { in = new DataInputStream(i); }
 public void run(){
   while(true){
    try{
       byte num = in.readByte();
       System.out.println(num);
    }catch(IOException e)
     { e.printStackTrace(); }
   }
 }
}// Fin clase ByteReader

// Escribe bytes en un stream de salida
class ByteGenerator extends Thread{
 private DataOutputStream out;
 byte b;

 public ByteGenerator(OutputStream o)
 { out = new DataOutputStream(o); }
 public void run(){
   while(true){
     try{
        b = (byte)(Math.random() * 255);
        out.writeByte(b); out.flush();
        sleep(500);
     }catch(IOException e)
      { e.printStackTrace();
     }catch(InterruptedException e)
      { e.printStackTrace(); }
   }
 }
}// Fin clase ByteGenerator

public class PipeTest{
 public static void main(String[] args){
   try{
     PipedOutputStream producer =
                    new PipedOutputStream();
     PipedInputStream consumer =
                    // Se realiza la conexión
                    new PipedInputStream(producer);
                    // Creo los threads
     ByteReader basket =
                    new ByteReader(consumer);
     ByteGenerator gen =
                    new ByteGenerator(producer);
                    // Arranco los threads
     gen.start(); basket.start();
   }
    catch(IOException e)
    { e.printStackTrace(); }
 }
}
```

## LineNumberInputStream

En el ejemplo que sigue leemos desde la entrada estándar e imprimimos cada línea con su número. Debemos recordar que la numeración comienza a partir del cero.

```
// PrtLines.java

import java.io.*;

public class PrtLines {
  public static void main(String[] args){
    // El contador de líneas
       LineNumberInputStream lnCount =new
            LineNumberInputStream(System.in);
    // Filtro para leer líneas
    DataInputStream inStr = new
            DataInputStream(lnCount);
       try{
          while(true){
            String nextLine = inStr.readLine();
            if (nextLine == null) break;
       System.out.print(

    lnCount.getLineNumber()+": ");
       System.out.println(nextLine);
          }
       }catch(Exception e)
          {
            e.printStackTrace();
          }
  }
}
```

## PushbackInputStream

Este *stream* nos permite recoger un simple carácter de un flujo de entrada y devolverlo de nuevo a dicha corriente de bytes (*pushback*), mediante métodos tipo *unread()*, para que pueda ser procesado de nuevo. Esta técnica es, a menudo, utilizada en la creación de *scanners* léxicos en la construcción de compiladores. La función mostrada bajo estas líneas, extrae las palabras clave encerradas entre los símbolos << y >>, a partir de un *array* de *bytes*, suministrado como argumento.

```
static public void findKeys(byte[] buff)
{ PushbackInputStream pist = new
    PushbackInputStream(new
      ByteArrayInputStream(buff));
 boolean bKey = false;
 String sKey = "";
 int c, c1;
 while(true){
       try{
          c = pist.read();
          if(c < 0) break;
          switch(c){
            case (int) '<':
              c1 = pist.read();
              if(c1 == (int) '<'){
                       bKey = true;
                       sKey = "";
              }else pist.unread(c1);
              break;
       case (int) '>':
                    c1 = pist.read();
                    if(c1 == (int)'>'){
                      bKey = false;
                      System.out.println(

       "Key: "+sKey);
                    }else pist.unread(c1);
                    break;

            default:
                    if(bKey) sKey += (char)c;
                    break;
          }
       }catch(IOException e){ break; }
 }// Fin del while
 try{ pist.close(); }
 catch(IOException e){}
}
```

## FilterInputStream

En el código de programa que aparece en el listado 1 podemos observar cuan simple puede llegar a resultar extender la funcionalidad básica que es ofrecida por los filtros de la biblioteca estándar. En este caso, el filtro construido elimina los espacios, cuando realiza sus operaciones de lectura sobre un *stream* de entrada.

## Tipos de OutputStream

La tabla 3 puede servirnos como ilustración de las diferentes subclases de *OutputStream*.

Análogamente a lo visto para los *streams* de entrada, la tabla 4 nos muestra la forma en que podemos mejorar la escritura en un *stream*, mediante la adición de filtros.

*PipedInputStream y PipedOutputStream ilustran el concepto de pipe*

**Listado 1**

```
// Prueba.java
import java.io.*;
// Una clase filtro que elimina los
// espacios de un fichero leído
class SkipWhiteFilterInputStream
                extends FilterInputStream {
  public SkipWhiteFilterInputStream(InputStream in){
    // Iicialización componentes de la clase base
super(in);
  }
  public synchronized int read() throws IOException {
    int c;
          do{
             c = in.read();
       if (Character.isSpace((char)c)){
                    continue;
             }else return c;
          }while(true);
  }
  public synchronized int read(byte[] b, int off, int len)
     throws IOException {
    byte[] tmp = new byte[len];
    int howmany = in.read(tmp);
    int real_count = off;
    for (int i = 0; i < howmany; i++) {
    if(!Character.isSpace((char)tmp[i]))
            b[real_count++] = tmp[i];
    }
    return (real_count);
  }
  public synchronized int read(byte[] b) throws IOException {
     return (this.read(b, 0, b.length));
  }
  public synchronized long skip(long n) throws IOException {
     byte[] b = new byte[(int)n];
     return (this.read(b));
  }
  public synchronized int available() throws IOException {
     int n = in.available();
          byte[] b = new byte[n];
          return this.read(b, 0, n);
  }
  public boolean markSupported() {
     return in.markSupported();
  }
  public synchronized void mark(int readlimit) {
     in.mark(readlimit);
  }
  public synchronized void reset() throws IOException {
     in.reset();
  }
  public synchronized void close() throws IOException {
     in.close();
  }
}
// Para comprobar su funcionamiento
public class Prueba {
  public static void main(String[] args) {
    if (args.length != 1) {
      System.err.println("Uso: java Prueba <input_file>");
      System.exit(-1);
    }
    try {
      FileInputStream f1 = new FileInputStream(args[0]);
           SkipWhiteFilterInputStream s1 = new
           SkipWhiteFilterInputStream(f1);
  // Leemos carácter a carácter
      for(int c = s1.read(); c > -1; c = s1.read()) {
        System.out.print((char)c);
      }
      s1.close(); // Cerramos el stream
  // Probamos check, mark y skip
      f1 = new FileInputStream(args[0]);
      s1 = new SkipWhiteFilterInputStream(f1);
      byte[] buf = new byte[100];
      int howmany;
           while ((howmany = s1.read(buf)) > 0) {
        for (int i = 0; i < howmany; i++)
          System.out.print((char)buf[i]);
        if (s1.skip(100) != 100)   break;
        if (s1.markSupported()) {
          s1.mark(100);
          if (s1.skip(100) != 100)
            break;
          s1.reset();
        }
      }
      s1.close();
    }catch(IOException e)
  {  System.out.println(e);  }
  }
} // Fin de la clase Prueba
```

Debido a la similitud en la forma de trabajar con las clases derivadas de *OutputStream,* en relación con las de *InputStream,* he decidido no incluir ningún ejemplo de código dentro de este apartado.

**Tabla 3 Subclases de OutputStream.**

| Clase | Función |
|---|---|
| ByteArrayOutputStream | Crea un *buffer* en memoria. Todos los datos enviados a dicho *stream*, son acomodados en dicho buffer. |
| FileOutputStream | Para enviar información a un fichero. |
| PipedOutputStream | La homóloga de *PipedInputStream.* |
| FilterOutputStream | La pareja de *FilterInputStream.* |

**Tabla 4. Filtros para OutpuStream.**

| Clase | Función |
|---|---|
| DataOutputStream | Utilizada en conjunción con *DataInputStream*, para poder escribir los tipos primitivos con los que cuenta el lenguaje. |
| BufferedOutputStream | Usado para prevenir una escritura física, cada vez que enviamos un cierto dato. |
| PrintStream | Utilizada para producir salida formateada. |

El proceso a seguir es muy sencillo: simplemente, en vez de leer de un *stream,* procederemos a escribir en él. La técnica de utilización de las diferentes clases mantiene de esta forma la misma tónica: encadenar distintas clases hasta que consigamos obtener la funcionalidad deseada.

## La clase file

Como primer ejemplo ilustrativo de las posibilidades de la clase file, he incluido un programa que nos imprime, por pantalla, el listado de todos los ficheros y directorios (es decir de todos los objetos *File*), que se encuentren en un *path,* especificado a través de la línea de comandos.

Asimismo, podemos observar la utilización del *interface FilenameFilter,* que nos permite limitar la salida del programa, a objetos con una cierta extensión (el funcionamiento no es perfecto, ¿cuál de los osados lectores se atreve a hacer un clónico del famosísimo comando ls de Unix?).

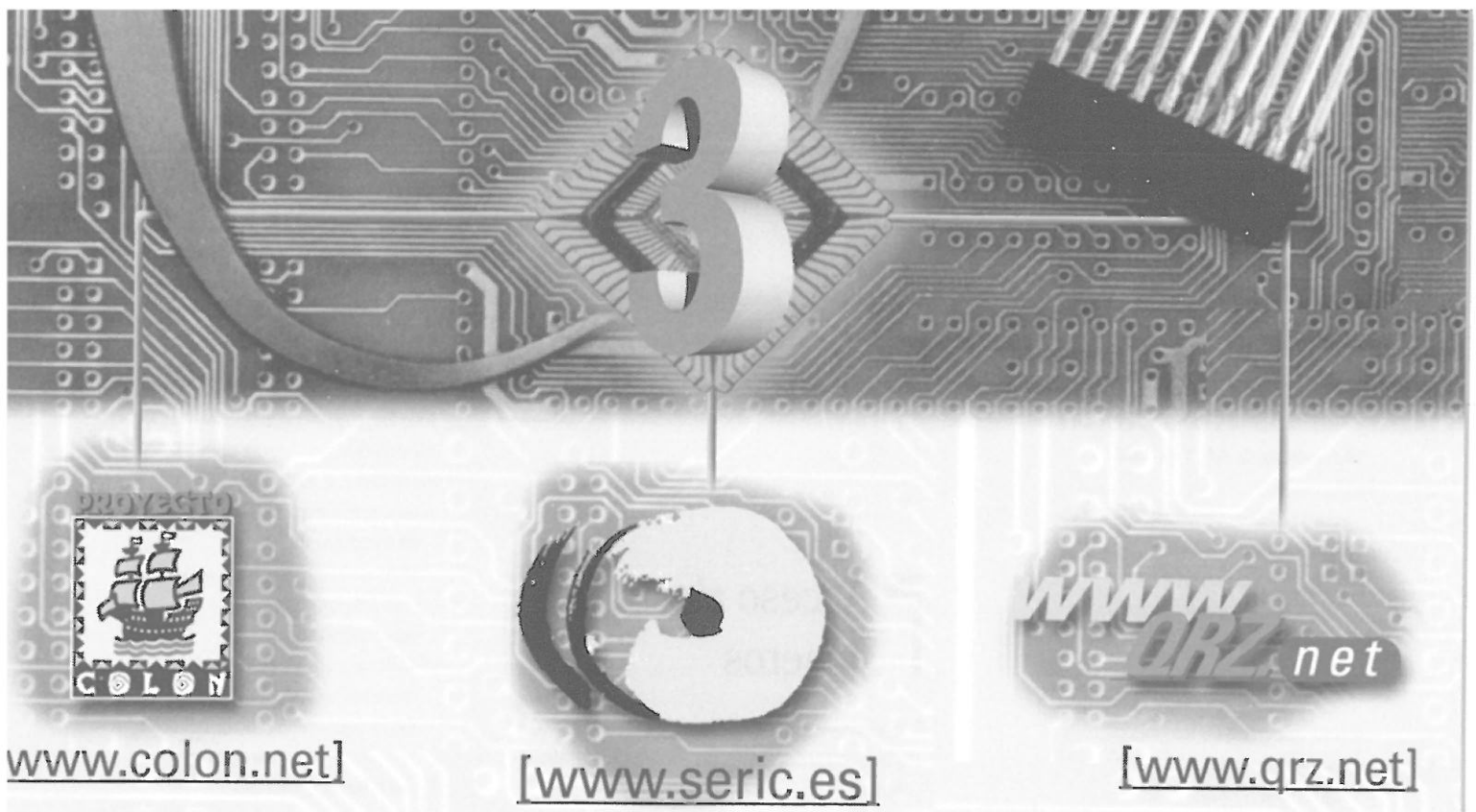
```
public interface FilenameFilter
{
  boolean accept(File dir, String name);
}
```

*El interface FilenameFilter nos permite limitar la salida del programa a objetos con una cierta extensión*

La tarea que se encarga de realizar este *interface* es bastante trivial. La función *list()* necesita en este momento acceder al patrón que debe ser tomado como filtro, y para ello efectuará una llamada a un método *accept(),* que devolverá entonces *true* o *false*, dependiendo de la aparición o no de dicho patrón en el nombre del objeto *File* que se haya procesado.

```
// Ls.java : Un DIR muy sencillo
import java.io.*;

public class Ls
{
 public static void main(String args[]){
       try
       {
         File path;  String[] list;
         if(args.length == 1)
       {
             path = new File(args[0]);
             list = path.list();
        }
       else if(args.length == 2)
       {
             path = new File(args[0]);
             list = path.list(new Filt(args[1]));
       }else{
             System.out.println("Error");
             return;
       }
         for(int i=0;i<list.length;i++)
             System.out.println(list[i]);
  }catch(Exception e)
       {e.printStackTrace();}
 }
}// Fin de la clase Ls

// Un sencillo filtro para las extensiones
class Filt implements FilenameFilter{
 String afn; final int LEN;
 Filt(String afn)
 {
    this.afn = afn;
   LEN = afn.length();
 }
 public boolean accept(File dir, String name){
   String f = new File(name).getName();
       int i;
   return ((i=f.lastIndexOf(afn)) != -1)&&
                        (i == (f.length()-LEN));
 }
}
```

Como segunda muestra de las virtudes y defectos que demuestra esta clase, vamos a estudiar sus habilidades en relación con la creación, borrado y renombrado de objetos *File,* así como los distintos métodos informativos con los que cuenta.



www.colon.net] [www.seric.es] [www.qrz.net]

# Disponemos de tres Servidores de INTERNET

**Seric**, empresa consolidada desde 1987, con amplia experiencia en el sector de las comunicaciones le ofrece un amplio abanico de **soluciones** para la empresa y el usuario particular ajustándonos a sus necesidades

**TARIFAS COLON**

- Conexión a INTERNET hasta el 31 de Enero de 1998 (6 meses)
- Cuenta de correo electrónico
- Curso interactivo en CD-Rom

4.000 ptas. I.V.A. incluido

**TARIFAS**

- Línea R.T.B. 25.000 ptas. + I.V.A.
  - Cuenta de correo
  - 1 Mb para sus páginas Web
- Línea R.D.S.I. 75.000 ptas. + I.V.A.
  - Cuenta de correo
  - 2 Mb para sus páginas Web

**TARIFAS www.QRZ.net**

- Línea R.T.B. 15.000 ptas. + I.V.A.
  - Cuenta de correo
  - 2 Mb para sus páginas Web
- Línea R.D.S.I. 35.000 ptas. + I.V.A.
  - Cuenta de correo
  - 4 Mb para sus páginas Web

85.000 ptas. + I.V.A.
- Alta y creación de dominio
- 4 direcciones E-mail
- Espacio ilimitado en disco

Telefónica Consultor homologado

Seric le tramitará el cambio de RTB a RDSI, o le gestionará el alta de una nueva línea RDSI, consúltanos

**SeriC's Room**
A ROM WITH A VIEW

Passeig Prim, 27 baixos Tel.977/ 33 11 40 Fax.977/ 33 11 14 - 43202 Reus http://www.seric.es - E-Mail: ventas@mail.seric.e

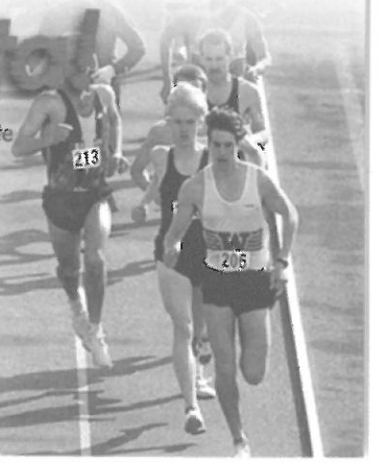
# stollmann ¡Velocidad Punta!

Para más información
http://www.stollmann.com

- Adaptador RDSI de alto rendimiento, fácil instalación, bajo coste y totalmente externo sin necesidad de manipular el equipo.
- Compatible con MS-DOS, Windows 3.X/95 NT, UNIX, MAC.
- TA+PPP Contiene todo lo que un adaptador RDSI necesita tener.

**Datos Trasmisión 115.000 bps.**

**32.000 ptas.** + I.V.A. Alimentador, manual, cable conexión serie y cable conexión línea RDSI.



Dado que el listado no ofrece ninguna complicación, dejo como tarea para el lector la comprensión del mismo y su experimentación.

```
//Mk.java

// Manipulación de fich. y directorios
import java.io.*;

public class Mk{
 private static void error(){
      System.out.println("Error");
      System.exit(-1);
 }

 private static void fileInfo(File f){
  System.out.println(
   f.getAbsolutePath() +" R: " +
   f.canRead() + " W: " + f.canWrite() +
   " " + f.getName() + " " +" " +
   f.getParent() + " " + f.getPath() +
   " " + f.length() + " " + f.lastModified());
 }

 public static void main(String args[]){
  if(args.length < 1) error();
     // renombrando objetos File
  if(args[0].equals("-r"))
  {
   if(args.length != 3)
          error();
   File  oldname = new File(args[1]),
    newname = new File(args[2]);
   oldname.renameTo(newname);
   fileInfo(oldname); fileInfo(newname);
   return;
  }

  // Borrando y creando objetos File
  int narg = 0;
  boolean del = false;
  if(args[0].equals("-d")){
   narg++;
   del = true;
  }

  // Podemos especificar cualquier cantidad
     for( ;narg< args.length;narg++){
   File f = new File(args[narg]);
   if(f.exists()&&del){
     System.out.println(" Eliminado "+f);
     f.delete();
   }
   else // Creamos directorio
    if(!f.exists()&&!del){
     f.mkdirs();
     System.out.println(" Creando "+f);
    }
   fileInfo(f);
  }
 }
}
```

# Acceso aleatorio a ficheros

En ocasiones no deseamos acceder de forma secuencial a un fichero (pasito a pasito).

La clase *RandomAccesFile*, nos permite leer y/o escribir bytes, en un fichero arbitrario, en una localización específica. Para ello cuenta con los métodos *getFilePointer()*, para encontrar la posición actual en el fichero; *seek()*, para saltar a un nuevo punto del fichero; y *length()*, que nos informa del tamaño del fichero (en *bytes*).

Cuando creamos un objeto de esta clase, suministramos como segundo argumento del constructor el modo de apertura. Especificamos dicho modo mediante una *String*, que puede tomar los valores "r", para sólo lectura o "rw", para lectura/escritura.

En el siguiente listado podemos observar el uso de sus métodos más corrientes.

```
// RAFTest.java

import java.io.*;

public class RAFTest{
 public static void main(String[] args){
  if(args.length != 1){
     System.err.println(
       "Uso: java RAFTest <fich.>");
     System.exit(-1);
  }
  try{
     RandomAccessFile raf = new
     RandomAccessFile(args[0], "rw");
     char a = 'a';
     byte b = 2;
     String str = "Java\n";
     byte[] b2 = {'a','e','i'};
     // Posición actual dentro del fich.
     long file_start = raf.getFilePointer();
     raf.write(b);
     raf.write(b2, 0, b2.length);
     // escribe el caracter con dos bytes
       raf.writeChar(a);
     // Sólo escribimos los 8 bits mas bajos
       // de cada caracter
       raf.writeBytes(str);
     // Cada caracter 2 bytes
       raf.writeChars(str);
       System.out.println("Long. del fich.: "+
                    raf.length());
       // Volvemos al principio
       raf.seek(file_start);
       b = raf.readByte();
       System.out.println("Byte: "+b);
       // Movemos el puntero
       raf.skipBytes(b2.length);
       raf.readChar(); b2 = new byte[5];
       // Leemos hasta llenar b2
       raf.readFully(b2);
       System.out.println("readFully: "+
          (char)b2[0]+(char)b2[1]+(char)b2[2]+
          (char)b2[3]+(char)b2[4]);
     // Leemos una línea
     StringBuffer sbuf = new StringBuffer();
       char  c;
       while ((c = raf.readChar()) != '\n')
            sbuf.append(c);
       System.out.println(sbuf.toString());
       // Cerramos el fichero
       raf.close();
  }catch(IOException e)
   {
       System.err.println(e);
   }
  }
}
```

# Los nuevos Streams

Después de todas las clases vistas, y sus correspondientes mezclas, uno se pregunta, ¿qué clases pueden haber sido

añadidas con el propósito de manejar *streams*, a partir de la versión 1.1 del Java Development Kit?

## *Debemos adoptar la jerarquía Reader/Writer que se utiliza a partir de la versión 1.1 del JDK*

A primera vista, la respuesta puede parecer desalentadora. En este momento debemos olvidarnos de la jerarquía *InputStream/OutputStream*, para adoptar la jerarquía *Reader/Writer.*

El cambio es debido fundamentalmente a que las antiguas clases estaban orientadas a *streams* de *bytes,* por lo que no encajaban con los caracteres Unicode, que comprenden 16 bits. Bajo

Tabla 5. Correspondencia entre clases.

| Antiguas jerarquías | Nueva jerarquía |
|---|---|
| InputStream | Reader |
| OutputStream | Writer |
| FileInputStream | FileReader |
| FileOutputStream | FileWriter |
| StringBufferInputStream | StringReader |
| (No existe similar) | StringWriter |
| ByteArrayInputStream | CharArrayReader |
| ByteArrayOutputStream | CharArrayWriter |
| PipedInputStream | PipedReader |
| PipedOutputStream | PipedWriter |
| FilterInputStream | FilterReader |
| FilterOutputStream | FilterWriter |
| BufferedInputStream | BufferedReader |
| PrintStream | PrintWriter |
| LineNumberInputStream | LineNumberReader |
| PushBackInputStream | PushBackReader |

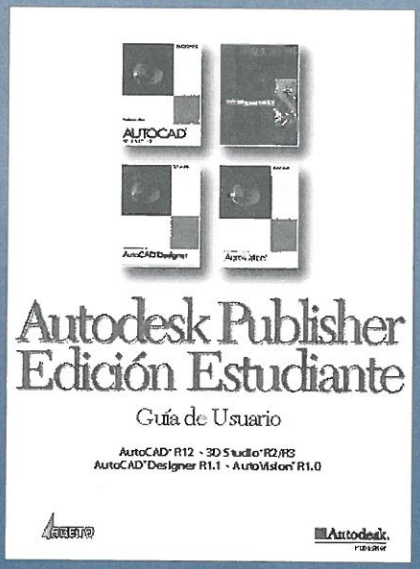

AUTODESK PUBLISHER EDICIÓN ESTUDIANTE

¿QUÉ PROGRAMAS INCLUYE?

- AutoCAD R12
- 3D Studio R2/R3
- AutoCAD Designer R1.1
- Autovision R1.0.

Y además:

Más de 700 páginas de lecciones de Autodesk con guías de referencia y documentación software on-line




TODO LO QUE NECESITA PARA EL DISEÑO Y VISUALIZACIÓN PROFESIONAL EN 2D Y 3D LO HALLARÁ EN ESTE PAQUETE DE PROGRAMAS.

Formato 21x29,7

EDICIÓN ESPECIAL PARA ESTUDIANTES
19.500 PTAS. (I.V.A. INCLUIDO)

DE VENTA EN LIBRERÍAS Y DISTRIBUIDORES AUTODESK

la maravillosa promesa de la internacionalización, estas nuevas jerarquías han sido diseñadas para soportar *streams* de caracteres, con lo cual Unicode ya es manejado en todas las operaciones de E/S.

Como valor añadido, las nuevas clases realizan su trabajo de manera más rápida que las antiguas debido, sobre todo, a que utilizan *buffers* internamente.

*Por suerte, las nuevas clases que incorpora el JDK 1.1 se corresponden casi exactamente con las antiguas*

Pero, por suerte, no todo está perdido. Como podemos observar en la Tabla 5, las nuevas clases se corresponden casi exactamente con las antiguas, respetándose también, de una manera bastante aceptable, las interfaces de las mismas.

Además, también afortunadamente, las cosas no son como en Fahrenheit 451 y los diseñadores de Javasoft indultaron del fuego a algunas de las antiguas clases. Las clases agraciadas fueron las siguientes:

- *DataInputStream*
- *DataOutputStream*
- *File*
- *RandomAccesFile*
- *SequenceInputStream*

Si observamos con detenimiento la lista anterior podemos deducir que existe en ella una pequeña incongruencia. En el supuesto de que deseemos guardar y recuperar los tipos básicos en un formato que sea portable entre distintas plataformas (mediante los *DataInputStream* y *DataOutputStream)*, nos veremos forzados a trabajar con las clases "*deprecated*".

Un pequeño detalle que comentar se refiere a la función *readLine()*. Deberemos usar *BufferedReader,* en vez de *DataInputStream*, si deseamos dicha funcionalidad con objeto de no tener ningún tipo de problemas con los caracteres Unicode.

En ciertas ocasiones, deberemos utilizar clases de ambas jerarquías en un mismo código. Para poder realizar dicha tarea contamos con dos clases puente: *InputStreamReader*, que convierte un *InputStream* en un *Reader;* y *OutputStreamWriter,* que transforma un *OutputStream* en un *Writer.*

Por ejemplo:

```
// Leemos de la entrada estándar

BufferedReader stdin =
      new BufferedReader(
            new InputStreamReader(System.in));
System.out.println("Introduce una línea");
System.out.println(stdin.readLine());
```

Puesto que la migración de unas jerarquías a otras es un proceso que puede parecer relativamente trivial, dejo como ejercicio para probar la capacidad del lector el intentar adaptar los ejemplos mostrados en este artículo para que utilicen las nuevas clases.

## Conclusión

A lo largo de este artículo hemos pretendido dar la vuelta al mundo de la entrada/salida en la programación en Java. Sin embargo, como en todo viaje que se precie, han quedado muchos lugares sin visitar que animo al lector a conocer por si mismo.

Aún así, espero que el lector haya perdido el miedo al ingente número de clases con que cuenta este *package de Java*.

El proceso de trabajo que se sigue en este tipo de operaciones siempre es similar: determinamos un *stream,* o un combinado de ellos, para lograr unas ciertas especificaciones, y a partir de entonces sólo nos queda escribir o leer sobre él.

*Un consejo: hay que intentar utilizar siempre que sea posible la jerarquía Reader/Writer, en vez de las clases antiguas*

Para terminar con el artículo que nos ocupa, me atrevería a dar un último consejo: sería muy recomendable intentar utilizar, siempre que nos fuera posible, la jerarquía *Reader/Writer*, en lugar de las antiguas clases. Únicamente de esta forma nos liberaremos de los problemas entre caracteres y *bytes*, que inundan al código que podíamos crear anteriormente a la aparición del JDK 1.1.

## Contactar con el autor

Para expresar cualquier duda que se presente o realizar cualquier comentario, sugerencia o crítica sobre éste u otros artículos de la serie que desde Sólo Programadores dedicamos a la programación en el lenguaje Java, se anima al lector a ponerse en contacto con el autor, Óscar Prieto Blanco, a través de la siguiente dirección de correo electrónico:

E-Mail myoprieto@redestb.es

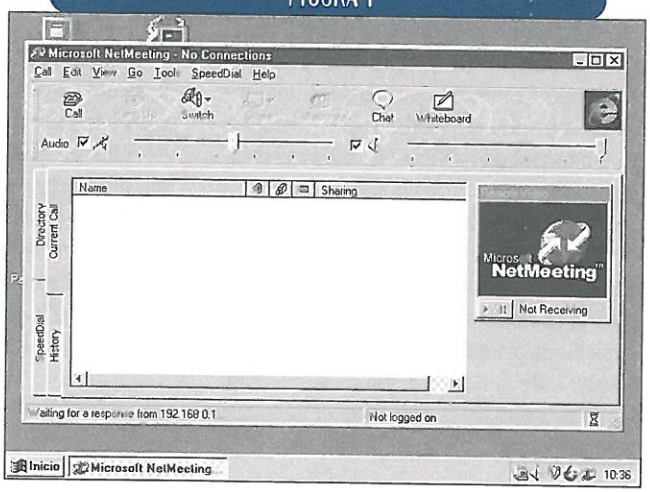
# Service packs NT 4.0 y Visual Studio, RealAudio 4.01 y así hasta 600 M de productos fundamentales

CONTENIDO DEL CD

En el número de Sólo Programadores que ahora tenéis entre vuestras manos nos atrevemos a retaros, si sois capaces de utilizar todo el software que incluimos en el CD-ROM, nos habréis vencido. El CD viene cargado de un gran número de utilidades y programas orientados a la programación de aplicaciones Internet/Intranet, también hemos incluido a petición de varios lectores de la revista versiones específicas de ciertas herramientas de desarrollo, ya incluidas en meses anteriores pero que este mes las incorporamos para el sistema operativo OS/2 de IBM. Así mismo, los usuarios de Windows NT 4.0 observarán con justificada gratitud que el presente CDROM contiene los últimos Service Packs o conjunto de parches para este sistema operativo, tanto para la versión en Castellano como para la Inglesa.

Y como no, versiones para Linux de plug-ins tan conocidos como el RealAudio y otras aplicaciones han sido incluidas también, con vista a ese grupo de expertos en el sistema operativo multitarea/multiusuario por excelencia, el Linux.

FIGURA 1



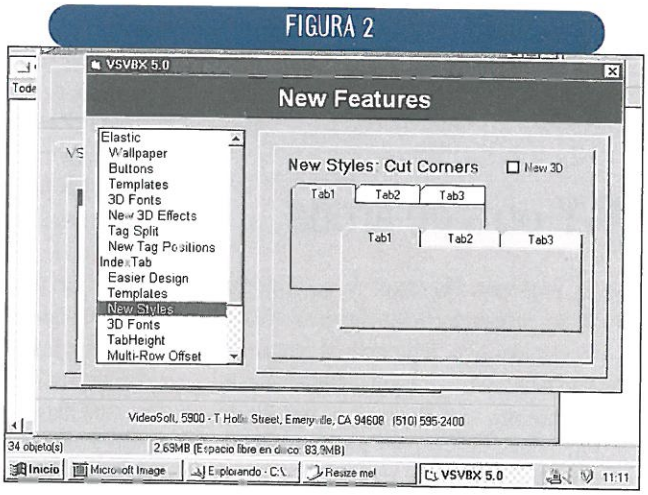

Y pasamos ahora a los usuarios del OS/2 Warp de IBM, ya que como adelantábamos hace un par de párrafos, este mes están de enhorabuena, debido a que podrán hacer uso de sus dotes de programación en Java mediante la versión 1.1.1 del Java Development Kit que incluimos.

## Internet

Y empezamos con la estrella de este número, la red Internet y sus numerosos protocolos y formatos multimedia. En primer lugar tenemos la versión 4.01 del RealAudio para Windows 95 & NT 4.0, o la versión 3.01 en su homólogo Linux, tanto en el novedoso formato *elf* como en el antiguo *a.out*. Éstos últimos usuarios encontrarán como cada mes en el directorio LSM, un documento de texto con la relación de los últimos programas y sus actuales versiones disponibles para este S.O.

Asímismo incluimos como novedad un mirror completo del site de Internet del Proyecto Lucas, que para aquellos que aún no sepan qué es exactamente, diremos que se trata de un proyecto que se está llevando a cabo para traducir los manuales y HOWTOs del Linux al Castellano.

FIGURA 2



Pero eso no es todo, recientemente salida al mercado, los lectores podrán evaluar las capacidades y características de la herramienta de desarrollo Visual Age for Java en su versión OS/2 y Windows 95/NT, la cual pasaremos a analizar profundamente en el próximo número de nuestra revista.

## Netscape Communicator 4.02

De Netscape también nos hemos ocupado, incluimos la última versión del Netscape Communicator, la 4.02, que incluye la tecnología Push. Sobre Netcaster y esta tecnología podéis encontrar más información en el artículo sobre WWW. En relación con este producto debemos incluir la siguiente información: la versión de Netscape Communicator se autoriza para la utilización offline y online, por un periodo de prueba de 60 días a partir de la fecha de publicación de Sólo Programadores y la aparición del CD que la acompaña.

Netscape no proporcionará soporte técnico del producto de este CD.

Para obtener una versión con licencia del software de Netscape Navigator, con o sin una suscripción a un proveedor de Internet, por favor contactar con APTIVA 091 364 0882, o con ADD 093 5802500.

## Y otra de arena

De la empresa Microsoft y su entorno Windows, proporcionamos diversas utilidades orientadas a la red como el fantástico NetMeeting 2.0 (castellano e inglés) con el que podremos conectarnos a través de TCP/IP con otras personas y comunicarnos ya sea escribiendo por chat, dibujando en una pizarra, hablando como si utilizásemos un teléfono, o incluso aquellos agraciados con una buena conexión y un potente hardware, por vídeo. En esta última revisión que ha lanzado Microsoft, se nos brinda la posibilidad de compartir aplicaciones remotamente, permitiendo hasta el control de éstas desde ambas partes.

De Microsoft también, se incluye la última Beta del novedoso Internet Explorer 4.0, que nos permitirá apreciar las diferencias entre las tecnologías Push/Pull de obtención de información, cuya versión final estará muy pronto, si no lo está ya en el momento de que estas líneas sean leídas por nuestro lectores.

Y como complemento ideal del Internet Explorer, para la consulta del correo y trabajo con los grupos de discusión de internet, para tener acceso a tus temas de programación favoritos o de lo que quieras, encontrarás también el Internet Mail & News de Microsoft. Son totalmente compatibles con el Microsoft OutLook de Office 97. Y, como no, disponibles las versiones en Inglés y Castellano.

## Software para Windows

Hasta los grandes cometen errores y un sistema operativo tan completo como éste necesita una continua revisión, actualmente van por la tercera en su modelo NT. Estos paquetes corrigen los "bugs" (errores) que los usuarios expertos del sistema operativo han ido encontrando al usarlo, además incluyen mejoras en algunos de los programas y librerías dinámicas que se adjunta siguiendo las recomendaciones de los expertos. Es necesario instalar la versión correspondiente con nuestro sistema operativo ya sea en el idioma inglés o castellano correspondiente. Las instrucciones de instalación son tan sencillas como ejecutar el programa setup y seguir el riguroso orden de salida de los Service Pack. La razón de incluirlos es la de ahorraros la necesidad de perder mucho tiempo conectados, ya que cada paquete ocupa alrededor de unos 15Mb.

## DirectX SDK

DirectX fue desarrollado para proporcionar a las aplicaciones Microsoft Windows un acceso de alta calidad y en tiempo real al hardware disponible. Incluyendo una interfaz consistente entre los fabricantes de hardware y el programador, reduciendo la complejidad de la instalación y configuración utilizando el hardware lo mejor posible. Incluimos también todo el software para el desarrollo de aplicaciones en este entorno.

## ActiveMovie

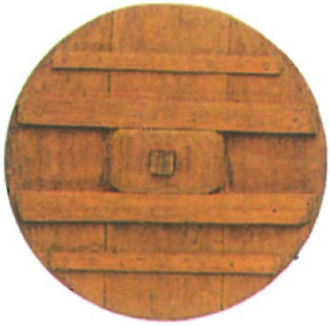
ActiveMovie es una arquitectura que controla y procesa streams de contenido multimedia; también es un ejecutable que usa esta arquitectura para permitir a los usuarios reproducir películas digitales y sonidos codificados en varios formatos incluyendo MPEG-1.

## Índice y Fuentes

Y cómo no, el índice de la revista y las fuentes de los artículos comentados en este número, fundamentales para sacar el máximo provecho y aprendizaje a las enseñanzas de nuestros excelentes colaboradores y redactores.

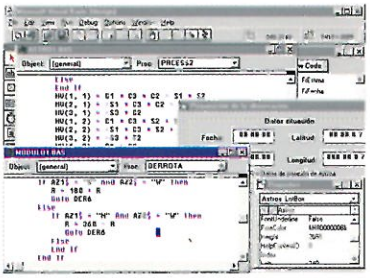
Nos despedimos así hasta el próximo mes, deseando saquéis el mayor provecho a ésta vuestra revista. Y, por supuesto, no dejéis de mandarnos sugerencias de todo aquellos que os gustaría ver incluido en el CDROM de próximos números, ya sabéis que haremos todo lo posible para que así sea.

# <herramientas que han ido cambiando el mundo>



<la rueda>

<la máquina de coser>



<Visual Basic /
Visual C++ / Visual J++>

<control de versiones y gestión
de configuraciones de software>

# <visual enabler>

## <una herramienta revolucionaria>

Microsoft Enterprise Development Partner

softlab

A BMW Company

VISUAL ENABLER

Softlab S.A. Puerto de los Leones, 1 - 28220 Majadahonda (Madrid) - Tel.: (91) 639 27 17 Fax.: (91) 639 31 55 - e-mail: softlab@ran.es - www.softlabna.com